ISSN 1807-4545

INFOCOMP

Journal of Computer Science

Endereço/*Address*
INFOCOMP – Journal of Computer Science
Departamento de Ciência da Computação
Universidade Federal de Lavras
Caixa Postal 3037
37200-000 – Lavras, MG, Brasil
Tel./Fax: +55 35 3829-1545
E-mail: infocomp@dcc.ufla.br
http://www.dcc.ufla.br/infocomp

Revista financiada com recursos da/*Journal financed with funds from*

FAPEMIG

Fundação de Amparo à Pesquisa do Estado de Minas Gerais

UFLA
UNIVERSIDADE FEDERAL DE LAVRAS

Ministério da Educação
Ministro: Fernando Haddad
Reitor: Antônio Nazareno Guimarães Mendes
Vice-Reitor: José Roberto Soares Scolforo
Pró-Reitora de Pesquisa: Édila Vilela de Resende Von Pinho
Editora UFLA
Presidente do Conselho Editorial: Renato Paiva
Volume 10, no. 2, June of 2011.

Indexed in: **INSPEC**; **Qualis-CAPES**.

INFOCOMP – Journal of Computer Science – v.10, n.2 (2011) – Lavras: Universidade Federal de Lavras, 2011.
Anual (1999 - 2003), Semestral (2004), Trimestral (2005 -)

Sumários em Inglês

ISSN 1807–4545

1. Ciência da Computação I.Universidade Federal de Lavras. II. Departamento de Ciência da Computação.

Solicita-se permuta /*Exchange desired* **Tiragem** /*Quantity issued per print*: 310.

A survey of point insertion techniques in bidimensional Delaunay Triangulations

S. L. G. DE OLIVEIRA

UFLA - Universidade Federal de Lavras
DCC - Departamento de Ciência da Computação
P.O. Box 3037 - Campus da UFLA 37200-000 - Lavras (MG) - Brazil
sanderson@dcc.ufla.br

Abstract. Triangulations are geometric discretizations essential in many scientific applications, such as engineering simulations, visualizations, and geographic information systems. The preferred shape of a triangle depends on the applications. Theoretical and experimental analysis of numerical methods that are used in conjunction with triangulations suggest that triangles with no large angles and/or small angles serve well in most applications. This paper is a brief review of a point insertion in 2D Delaunay Triangulations. Important works on the insertion of vertices in Delaunay Triangulations are described as a start point for one who needs to build a quality mesh using adaptive triangular-mesh refinement.

Keywords: Delaunay Triangulation, mesh generation, adaptive triangular mesh refinement, computational geometric modeling.

(Received April 10th, 2011 / Accepted June 18th, 2011)

1 Introduction

Triangulations are geometric discretizations essential in many scientific applications, such as engineering simulations, medical imaging, visualizations, and geographic information systems [22]. Erten and Üngör [22] explain that the preferred shape of a triangle depends on the applications. However, theoretical and experimental analysis of numerical methods that are used in conjunction with triangulations suggest that triangles with no large angles and/or small angles serve well in most applications (see [1]). According to Erten and Üngör [22], "in general, the better the shape of the triangles, the smaller the interpolation and approximation errors are in their use".

Delaunay triangulations are optimum in maximizing the smallest angle [17]. An approach in order to provide quality triangular meshes is to use algorithms based on a automatic point insertion strategy on the Delaunay Triangulation. A planar Delaunay Triangulation [15] for a point set P is a triangulation DT(P) such that no point in P is inside the circumcircle of any triangle in DT(P). The Delaunay Triangulation builds the optimal triangular mesh. This means that it builds triangles more similar to the equilateral ones for a given fixed point set.

The Delaunay Triangulation and its duals Voronoi Diagram [52] and medial axes have been applied in many different fields, such as the ones earlier cited, including numerial methods and computer graphics. The reader is referred to Guibas and Stolfi [26] and Barth [4] for properties and algorithms in order to build 2D Delaunay Triangulations. Shewchuk [45] presented aspects of the Delaunay mesh generation. Edelsbrunner [18] provided a theoretical review on Delaunay Triangulation. De Floriani and collaborators [12] reviewed the basic triangulation properties, Delaunay Triangulations, constrained and conforming triangulations. They also presented a survey of algorithms for building these kind of triangulations, mainly in the context of digital terrain modeling in geographic information systems.

In order to build a Delaunay Triangulation, the reader is referred to the mesh generation software Tri-

angle [48]. Triangle's high-quality mesh generation is based on Chew-Ruppert Delaunay refinement algorithm [41]. Both were surveyed by Shewchuk in [46]. In addition, Shewchuk described Ruppert's Delaunay refinement algorithm in [47]. These algorithms evolved from the works of Chew [7] and Bern *et al*. [5]. The Chew-Ruppert Delaunay refinement method is modified in Triangle to handle domains with small angles well, following a idea in the paper of Miller *et al*. [32]. It also incorporates a modification by Üngör [51] that reduces the number of triangles generated. Triangle's implementation of the divide-and-conquer and incremental Delaunay triangulation algorithms follows closely the presentation of Guibas and Stolfi [26]. Triangle uses a triangle-based data structure instead of Guibas and Stolfi's quad-edge data structure. The $O(n \log n)$ divide-and-conquer algorithm promoted by Guibas and Stolfi was originally developed by Lee and Schachter [31]. Dwyer [16] showed that the algorithm is improved by using alternating vertical and horizontal cuts to divide the vertex set. Triangle uses an expected $O(n^{1/3})$ time point location scheme proposed by Mücke [33]. Triangle's $O(n \log n)$ sweepline algorithm for Delaunay triangulation is due to Fortune [23], and relies upon Sleator and Tarjan's splay trees [50]. The earlier description is based on the Triangle's website [48].

Given a Delaunay Triangulation, one is allowed to insert points (called the Steiner points) in order to compute good quality triangulations. This, however, may increase the number of points and triangles in a triangulation, which is a key factor in the running time of an application algorithm. The reader is referred to [22] for details and a survey on the context of providing a good triangulation.

After this brief introduction, Section 2 provides a further review of the schemes for point insertion in a Delaunay Triangulation in the context of providing a adaptive refined mesh. Section 3 describes the Voronoi Diagram. Section 4 surveys the Rivara's schemes and others. Some future directions are given in Section 5.

2 Point insertion in a Delaunay Triangulation

A point insertion in a Delaunay Triangulation is not a trivial task. For example, if one simply inserts a point into the triangle barycenter (Figure 1a), this process fastly degenerates the triangulation quality, specially along boundaries. This occurs even when carrying out global refinement. In [38], the authors affirm that a pure Delaunay algorithm does not provide a natural point insertion scheme that guarantees the construction of good-quality nonuniform triangulations when the algorithm is iteratively used in the adaptive mesh refinement. They described experiments with the simple centroid insertion (see Figure 1a) concept.

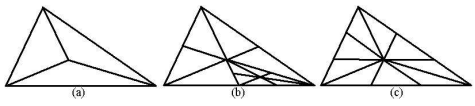

Figure 1: Triangle partition processes: (a) *ternary subdivision* - refinement by simple centroid insertion; (b) refinement by centroid insertion and adding midedges - a second refinement is performed in the bottom right triangle; (c) trisection of the edges, joinning the centroid to those points and also to the vertices.

The literature is rich in approaches to introduce points into the triangulation. These schemes provide high-quality Delaunay Triangulations and some of them are described in the following.

Fowler and Little [24] proposed the *vertex insertion* in conjunction with the Delaunay Triangulation. A Delaunay criterion localizes the position of a potential point to be inserted. This could affect the fit to the circumscribed circle about the triangle. The authors argued that it is sufficient to perform series of domain-limited searches in each triangle of the model; rather than carrying out global searches for the global "worst-fit" points. In this approach, adding a point destroys the original triangle and introduces new triangles. The inserted point is a vertex of the new triangles. In Figure 2, a point is introduced and the region is triangulated. The reverse operation, known in computer-graphic context as *decimation*, is performed in order to unrefine the region. In a variation, a point is inserted, the set of triangles on its neighborhood are deleted and the region is retriangulated (Figure 3). The inverse operator, the *vertex removal*, deletes a point together with its incident triangles and constructs new triangles in the region.

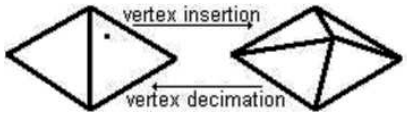

Figure 2: Vertex insertion and vertex decimation.

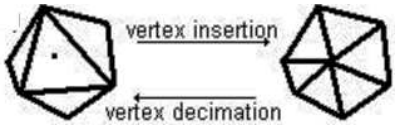

Figure 3: Vertex insertion and vertex decimation.

Clarkson and Shor [9] showed that if the order of vertex insertion is randomized, each vertex can be inserted in $O(n)$ time, not counting point location (see de-

tails in Shewchuk [48]. Chew ([7] and [8]) proposed a Delaunay improvement algorithm that triangulates a given polygon into a uniform mesh with all angles between 30 and 120. It guarantees that the output mesh is size-optimal within a constant factor amongst all uniform meshes.

The Hierarchical Delaunay Triangulations (HDT) was proposed by De Floriani and Puppo in [13] and [14]. It is based on a hierarchy of triangle-based surface approximations, where each node, except the root, is a triangulated irregular network refining a triangle face belonging to its parent in the hierarchy (see Figure 4). This method is similar to the proposed by Scarlatos Pavlidis in [42]; however, the triangle subdivision is more general. The subdivision inside every macrotriangle is locally a Delaunay Triangulation; whereas a global expanded subdivision of the whole domain gerally is not. The triangle partition is performed by an iterative application of a selector process that, at each step, updates the current Delaunay Triangulation by introducing the point having the maximum error. Moreover, in order to subdivide a triangle for a given hierarchical level, they used a curve approximation algorithm [3] in order to insert points along the edges. Afterwards, points are added in the inner triangle until an error threshold is met throughout the triangle. So, the inner triangle is retriangulated using Delaunay Triangulation. The constructing algorithm basis for a HDT must be an on-line approach that incrementally builds a Delaunay Triangulation through iterative point insertion [12]. According to Heckbert and Garland [28], the HDT seems to present nearly identical flexibility and speed compared to the one proposed in [42]. However, for a given error threshold, the HDT likely yields slightly better simplification.

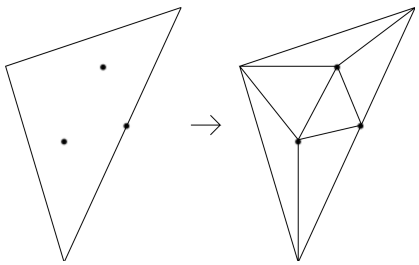

Figure 4: Hierarchical Delaunay Triangulations.

Ruppert [41] presented an algorithm to triangulate planar straightline graphs. It guarantees that every triangle in the output mesh has smallest angle greater than 278. It produces a size-optimal nonuniform mesh. It is also size-optimal to within a constant factor. The idea behing these algorithms is either: to refine a small angled triangle by the Delaunay insertion of its circumcenter; or a modification of the boundary if the circumcircle is external to the meshing region (see [37] for details). Baker [2] published a comparison of edge and circumcenter based refinements. Properties of mesh improvement for iterative Delaunay refinement based on inserting a point in the circumcenter of triangles to be refined was also established by Shewchuk in [44]. A combination of edge refinement and Delaunay point insertion was described by Borouchaki and George in [6] and [25].

Shewchuk [46] presentd a framework for analyzing Delaunay refinement algorithms that unifies the mesh generation algorithms of Chew and Ruppert. The Shewchuk's framework improves the Chew's and Ruppert's algorithms in several ways, and also helps to solve the difficult problem of meshing nonmanifold domains with small angles.

Üngör [51] presented an algorithm based on the *off-center* insertion. In the former case, the off-center of a triangle with the shortest edge $\overline{pq}$ is a point o on the bisector of $\overline{pq}$ furthest from p (or q) such that the angle among the three points is a user-specified constraint angle. The idea of using *off-centers* led Har-Peled and Ungor [27] to the design of the first time-optimal Delaunay refinement algorithm.

Erten and Üngör [21] proposed algorithms that improve the off-center performance with respect to the mesh size and a minimum angle tolerance. This is performed by using point selections depending on some triangle cases. Erten and Üngör [20] published a Delaunay refinement algorithm that generally terminates for constraint angles up to 42°.

Erten and Üngör [22] proposed two algorithms to improve the performance of Delaunay refinement. The first one uses the Voronoi Diagram and unifies previously suggested Steiner point insertion schemes (circuncenters [7], [40], [46], sink [19], off-center [51]) together with a proper strategy. The second algorithm integrates a local smoothing strategy into the refinement process. For a given input domain and a constraint angle α, the Delaunay refinement algorithms aim to compute triangulations with angles at least α.

Recently, Plaza and collaborators [34] proposed the 7-triangle Delaunay partition (Figure 5). This refinement scheme also propagates the refinement and inserts non-similar triangles.

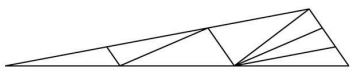

Figure 5: 7-triangle Delaunay partition

3 Voronoi Diagram

The Voronoi Diagram was proposed in [52]. Shamos [43] was the first to argue that the Voronoi Diagram can be used as a tool to provide efficient algorithms for a wide variety of geometric problems.

Barth [4] defined the Delaunay Triangulation of a point set as the dual of the Voronoi Diagram of the set. The 2D-Delaunay Triangulation is formed by connecting two points if and only if their Voronoi regions have a common border segment. If no four or more points are cocircular, then the vertices of the Voronoi Diagram are the triangle circumcenters. Moreover, Voronoi vertices represent locations that are equidistant to three or more points.

Consider the Delaunay Triangulation of a set *V* of planar points. The Voronoi Diagram describes the proximity relationship among the points of *V*. The Voronoi Diagram of a set *V* of *n* points is a planar subdivision into *n* convex polygonal regions. Each region is associated with a point of *V*. Each Voronoi region of each point of *V* is the set of planar points which lie closer to the point than to any other point in *V*. Two points of *V* are neighbors when the corresponding Voronoi regions are adjacent [12].

An interface orthogonal to the segment between two centroids facilitates finite-volume approximations. Moreover, it improves the solution accuracy and reduces the computational effort to approximate a solution of a partial differential equation. Furthermore, in this approach, the finite volumes are not the triangles themselves, but the Voronoi Diagram (see Figure 6), i.e. parts of each triangle.

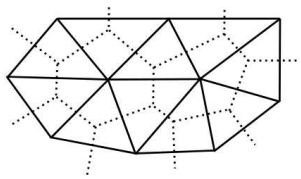

Figure 6: A single Delaunay Triangulation and its dual the Voronoi Diagram.

4 Longest-edge based triangle partition within Delaunay Triangulation

Rivara [35] presented the backward longest-edge refinement (BLER) algorithm based on an interesting concept in order to conform the mesh in the finite element context: the longest-edge propagation path (LEPP). Briefly, the LEPP keeps a path of *n* triangles that have also to be refined for each triangle of the mesh. For example, consider that the triangle t_0 is marked to be refined. The LEPP indicates that the triangles $t_1, t_2, \cdots, t_n$ also must be refined in order to mantain a conforming good-quality mesh. It propagates the list until the longest-edge shared by triangles t_{n-1} and t_n. This edge is larger than the one of its previous neighbor or t_n is in the boundary. Figure 7 shows an example of the LEPP-midedge propagation with 3T-LE partition approach and t_n is bisected, where *n*=4 in this example. The BLER is a partition procedure that extended both the pure longest-edge refinement algorithms for general nonDelaunay Triangulation (see [39] and the references therein) and the longest-edge refinement algorithm for Delaunay Triangulations proposed by Rivara and Inostroza [38]. Specifically, the algorithm presented in [38] guarantees that meshes of analogous quality to the input reference-mesh are built.

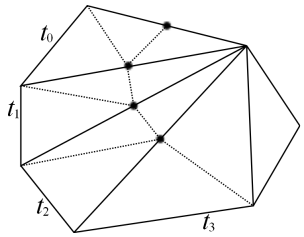

Figure 7: LEPP-midedge of t_0.

Rivara and collaborators ([37] and [49]) presented the LEPP-Delaunay midedge algorithm . It generalized and improved both previous longest-edge algorithms for the Rivara's refinement of general nonDelaunay Triangulations, and the longest-edge algorithm for the refinement of Delaunay meshes [38].

In the LEPP-Delaunay midedge algorithm, only considering local information associated to the terminal triangle that contains a constrained edge allows a real constrained Delaunay Triangulation. The constrained Delaunay Triangulation is the best approximation of the Delaunay Triangulation containing the set of given segments among its edges.

The LEPP-Delaunay midedge algorithm avoids the interaction with the entire set of constrained items. This algorithm is not a nested partition procedure because it changes the previously existing points. Moreover, it replaces previous triangles by Delaunay triangles due to the circumcircle test of DT(P). In addition, it suffers of a looping case for angle tolerance greater than 22°. Namely, in certain cases, the triangles are not improved during the refinement. Nevertheless, it is interesting since it provides meshes with triangles which the smallest angle is greater than or equal to $\pi/6$, including along boundaries.

Hitschfeld and Rivara [29] introduced a automatic

construction of nonobtuse triangles in boundary for LEPP-Delaunay Triangulations within control volume methods. Each 1-edge obtuse boundary triangle is eliminated by the Delaunay insertion of midedges.

Consider that α is the smallest angle of the triangle. In the case that $\alpha \geq 25.4°$, any isolated 1-edge obtuse triangle and isolated pairs of neighbour 1-edge obtuse triangles sharing their longest edge demand the insertion of only one point. When $\alpha \geq 15.4°$, the Delaunay insertion of at most three boundary/interface points eliminates any isolated 1-edge boundary triangle and isolated pairs of neighbour 1-edge boundary triangles sharing a longest edge. An obtuse angle in each isolated 2-edge boundary triangle having medium-size edge *l* and longest-size edge *L* over the boundary is eliminated by building an isosceles triangle of boundary edges of lengths *l*/2 followed by the Delaunay insertion of a finite number of points *N*, where $N \leq \frac{2.14}{sin(\alpha/2)}$.

A generalization of those approaches solves more complex patterns of obtuse triangles, i.e. chains of 2-edge constrained triangles forming a saw diagram and clusters of triangles that have boundary/interface edges sharing a common vertex [29]. Hitschfeld and collaborators [30] presented the LEPP algorithm for Delaunay mesh and its dual Voronoi Diagram, without obtuse angles opposite to the boundary and interfaces for semiconductor device simulation using Box-method Delaunay meshes.

Rivara and Calderon [36] presented the LEPP-Delaunay centroid algorithm. They proved that the centroid version of the LEPP-Delaunay algorithm produces triangulations both with average smallest angles greater than those obtained with the midedge version and with larger smallest edges without suffering from the looping case associated to the midedge method. In addition, the centroid version terminates for high-quality threshold angle, i.e. up to $\pi/5$. They also showed that the centroid version behaves better than the off-center algorithm for quality threshold angle larger than $25°$.

Because the finite-element conformity requirement, most of those previous articles describe algorithms that propagate the refinement in neighbors of the triangle marked to be refined and/or modify the points of the current mesh. As an example, Rivara and Inostroza [38] pointed out that numerical experiments performed with their 2D algorithm have shown that the number of points inserted by propagation is approximately $N^{1/2}$, where N is the number of points in the mesh.

If an algorithm modifies the positions of the refined-triangle points, the data-structure nodes that represent those triangles also have to be changed. A process that operates strict local changes (a nested mesh) is desirable. In [26] and [11], the authors described algorithms that perform the circumcircle test of DT(P) without locally destroying the current triangulation.

In [36], for constrained edges, in both the circumcenter and the off-center algorithm if a prospective point P to be inserted is inside the diameter circle of any constrained edge E, the midpoint of E is inserted instead of P. This implies that a strict Delaunay Triangulation is maintained. As a result, no angle lesser than $\pi/2$ appears in the triangulation.

5 Concluding remarks

Plaza and collaborators [34] provided several open problems related to their 7-triangle partition approaches. There is a lot of work related to 3D (for example, see [10]). In addition, the 3D review shall be provided.

The purpose of this article is to survey the approaches and not to evaluate them. Probably other schemes exist. However, such schemes may be either variations of the ones cited in this article or are not known to the author. However, the author hopes that this review and the references cited serve to consolidate the ideas, principles and schemes that constitute the state-of-art in this subject. Moreover, the author hopes that the list of references and descriptions to the large body of work on this issue can provide a useful starting point for one faced with the task of adaptively constructing a Delaunay Triangulation.

6 Acknowledgements

This work was supported by FAPEMIG under the project CEX-APQ-01198-10, year 2010.

References

[1] Babuzka, I. and Aziz, A. K. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13:214–226, 1976.

[2] Baker, T. J. Triangulations, mesh generation and point placement strategies. In Caughey, D., editor, *Computing the future. John Wiley*, pages 61–75, 1995.

[3] Ballard, D. H. and Brown, C. M. *Computer Vision*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[4] Barth, T. J. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes Equations. In *Von Karman Institute for Fluid Dynamics Lecture Series, NASA Ames Research Center, 1994-05*, February 1995.

[5] Bern, M., Eppstein, D., and Gilbert, J. R. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994.

[6] Borouchaki, H. and George, P. L. Aspects of 2-d Delaunay mesh generation. *International Journal for Numerical Methods in Engineering*, 40:1957–1975, 1997.

[7] Chew, L. P. Guaranteed-quality triangular meshes. Technical Report 983, Department of Computer Science, Cornell University, 1989.

[8] Chew, L. P. Constrained Delaunay triangulation. *Algorithmica*, 4:97–108, 1994.

[9] Clarkson, K. L. and Shor, P. W. Applications of random sampling in computational geometry ii. *Discrete & Computational Geometry*, 4(1):387–421, 1989.

[10] Danovaro, E., De Floriani, L., Magillo, P., Puppo, E., and Sobrero, D. Computer Graphics in Italy - Level-of-detail for data analysis and exploration: A historical overview and some new perspectives. *Computers & Graphics*, 30:334–344, 2006.

[11] De Floriani, L. Surface representation based on triangular grids. *The Visual Computer*, pages 27–48, 1987.

[12] De Floriani, L., Bussi, S., and Magillo, P. Triangle-based surface models. *Chapter 9 in Intelligent Systems and Robotics, Breach Science Publishers*, pages 340–373, 2000.

[13] De Floriani, L. and Puppo, E. A hierarchical triangle-based model for terrain description. In et al., A. U. F., editor, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Berlin, Springer-Verlag*, pages 236–251, 1992.

[14] De Floriani, L. and Puppo, E. Extrating contour lines from a hierarchical surfase model. *Computer Graphics Forum (Proceedings Eurographics 93)*, 12(3):249–260, 1993.

[15] Delaunay, B. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[16] Dwyer, R. A. A faster divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, 2(2):137–151, 1987.

[17] Edelsbrunner, H. Triangulations and meshes in computational geometry. *Acta Numerica*, 9:133–213, 2000.

[18] Edelsbrunner, H. *Geometry and Topology for Mesh Generation. In: Cambridge monographs on applied and computational mathematics*. Cambridge University Press, New York, 2001.

[19] Edelsbrunner, H. and Guoy, D. Sink insertion for mesh improvement. In *Proceedings of the 17th ACM Symposium on Computational Geometry*, pages 115–123, 2001.

[20] Erten, H. and Üngör, A. Computing acute and non-obtuse triangulations. In *Canadian Conference on Computational Geometry (CCCG)*, pages 205–208, 2007.

[21] Erten, H. and Üngör, A. Triangulations with locally optimal Steiner points. In Belyaev, A. and Garland, M., editors, *Eurographics Symposium on Geometry Processing*, pages 143–152, 2007.

[22] Erten, H. and Üngör, A. Quality triangulations with locally optimal steiner points. *SIAM Journal of Scientfic Computing*, 31(3):2103–2130, 2009.

[23] Fortune, S. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987.

[24] Fowler, R. and Little, J. Automatic extraction of irregular network digital terrain models. *ACM Computer Graphics (SIGGRAPH '79 Proceedings)*, 13(3):199–207, 1979.

[25] George, P. L. and Borouchaki, H. *Delaunay triangulation and meshing*. Hermes, 1998.

[26] Guibas, L. J. and Stolfi, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi Diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[27] Har-Peled, S. and Üngör, A. A time-optimal delaunay refinement algorithm in two dimensions. In *Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy*, pages 228–236, 2005.

[28] Heckbert, P. S. and Garland, M. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University - Department of Computer Science, 1 May 1997.

[29] Hitschfeld, N. and Rivara, M. C. Automatic construction of non-obtuse boundary and/or interface Delaunay triangulations for control volume methods. *International Journal for Numerical Methods in Engineering*, 55:803–816, 2002.

[30] Hitschfeld, N., Villablanca, L., Krause, J., and Rivara, M. C. Improving the quality of meshes for the simulation of semiconductor devices using LEPP-based algorithms. *International Journal for Numerical Methods in Engineering*, 58:333–347, 2003.

[31] Lee, D.-T. and Schachter, B. J. Two algorithms for constructing the delaunay triangulation. *International Journal of Computer and Information Science*, 9(3):219–242, 1980.

[32] Miller, G. L., Pav, S. E., and Walkington, N. J. When and why ruppert's algorithm works. In *Twelfth International Meshing Roundtable*, pages 91–102, September 2003.

[33] Mücke, E. P., Saias, I., and Zhu, B. Fast randomized point location without preprocessing in two- and three-dimensional delaunay triangulations. In *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, May 1996.

[34] Plaza, A., Márquez, A., Moreno-González, A., and Suárez, J. P. Local refinement based on the 7-triangle longest-edge partition. *Mathematics and Computers in Simulation*, 79:2444–2457, 2009.

[35] Rivara, M. C. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *International Journal for Numerical Methods in Engineering*, 40:3313–3324, 1997.

[36] Rivara, M. C. and Calderon, C. LEPP terminal centroid method for quality triangulation. *Computer-Aided Design*, 42:58–66, 2010.

[37] Rivara, M. C., Hitschfeld, N., and Simpson, B. Terminal-edges Delaunay (small-angle based) algorithm for the quality triangulation problem. *Computer-Aided Design*, 33:263–277, 2001.

[38] Rivara, M. C. and Inostroza, P. Using longest-side bisection techniques for the automatic refinement of Delaunay triangulations. *International Journal for Numerical Methods in Engineering*, 40:581–597, 1997.

[39] Rivara, M. C. and Venere, M. Cost analysis of the longest-side (triangle biection) refinement algorithms for triangulations. *Engineering with Computers*, 12:224–234, 1996.

[40] Ruppert, J. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proceedings of the Fourth ACM-SIAM Symposium on Discrete Algorithms*, pages 83–92, 1993.

[41] Ruppert, J. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, May 1995.

[42] Scarlatos, L. and Pavlidis, T. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[43] Shamos, M. I. *Computational geometry*. PhD thesis, Yale University, New Haven, Conn., 1977.

[44] Shewchuk, J. R. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *First Workshop on Applied Computational Geometry. ACM*, pages 124–133, 1996.

[45] Shewchuk, J. R. Lecture notes on Delaunay mesh generation, 1999.

[46] Shewchuk, J. R. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 22:21–74, 2002.

[47] Shewchuk, J. R. Ruppert's delaunay refinement algorithm. website, July 2005.

[48] Shewchuk, J. R. Triangle: A two-dimensional quality mesh generator and delaunay triangulator. website, July 2008.

[49] Simpson, B., Hitschfeld, N., and Rivara, M. C. Approximate shape quality mesh generation. *Engineering with Computers*, 17:287–298, 2001.

[50] Sleator, D. D. and Tarjan, R. E. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

[51] Üngör, A. Off-centers: A new type of Steiner points for computing size-optimal quality-guaranteed Delaunay triangulations. In *Proceedings of the Latin American Symposium on Theoretical Informatics, Buenos Aires, Argentina*, pages 152–161, April 2004.

[52] Voronoi, G. Nouvelles applications des paramétres continus à la théorie des formes quadratiques. *Journal fur die Reine und Angewandte Mathematik*, 133:97–178, 1907.

Operational Profiles for Statistical Testing of Distribution Management System

ILIJA BASICEVIC[1]
ILIJA KUPRESANIN[2]
MIROSLAV POPOVIC[1]

[1]University of Novi Sad
Faculty of Technical Sciences
21000 Novi Sad- Serbia
[2]JP SRBIJAGAS
Narodnog Fronta 12
21000 Novi Sad - Serbia
[1]ilibas@uns.ac.rs,miroslav.popovic@rt-rk.com
[2]ilija.kupresanin@srbijagas.com

Abstract. Each generation of software systems is becoming more complex. Also, software is becoming more important because today critical infrastructure systems depend on software. This paper presents the method applied in testing of a complex software system. For complex systems, it is very important to measure their reliability. Statistical testing based on operational profiles is a de facto industrial standard for this purpose. As a case study, we used Windows-based distribution management system that is used in electric power distribution utilities. A library that contains the analytical functions subsystem was tested. The paper gives an overview of the system and module being tested, and of the statistical method we applied. The main contribution of this paper is development of operational profiles for a real-world complex system. Two of the operational profiles we have developed for testing of the system are presented in detail. Another contribution is a new approach which supports generation of test cases on-the-fly: during execution of implementation under test on a test bed. This is possible by joining together the test case generator and the test bed.

Keywords: power distribution management system, statistical testing, operation profiles, software testing, software reliability.

(Received December 8th, 2010 / Accepted June 16th, 2011)

1 Introduction

Statistical testing is a technique that originates from quality assurance in automatic production of mechanical and electrical devices. Given the importance that complex software systems have in everyday life, sometimes in critical applications, it is no surprise that this technique has found its place in software engineering, too.

Along the time scale, software of today is getting more complex. The statistical testing using operational profiles is a method for measuring reliability of complex software systems. An example of critical infrastructure system that depends on software is Distribution Management System (DMS) in electrical power distribution network, the case study that is used in this paper. The complexity of this class of software systems stems from the complexity of mathematical models of distribution networks, which are the basis for the software model. As an example of complexity, the input space for DMS in the case of the State of Texas measures 13

million of variables [8].

This paper presents operational profiles we have developed for DMS system. In our approach, the test case generator and test bed are joined together. This way, system supports generation of test cases during execution of implementation under test on the test bed.

The DMS Software is a software package for performing technical tasks in electric power distribution utilities. This software tool enables utility's staff to design and manage the network's power and automation equipment in order to maximize the quality and quantity of the electrical energy supplied to the consumers.

The DMS software is modularly organized package with three-tier software architecture. It is based on standard software solutions that should allow for simple integration with other standard software and hardware equipment found in the environment of electricity distribution (SCADA Systems, equipment for medium voltage (MV) automation, etc.).

The general DMS software architecture is briefly described as follows. The first tier is a relational data base. The middle tier integrates static technical and historical data with dynamic data (available for example from third party SCADA systems). In the third tier, there are user clients - Microsoft Windows applications. In the actual software architecture, some of third tier clients represent a shell for the DMS analytical functions system.

The most important applications of the third tier are [2]:

- Front-end application for data base editing. Used for editing of parameters of network elements, of their connectivity and finally their graphic representation in the form of a network diagram.

- Multiple-view user interface for visualization of supply and distribution substations and MV and LV network, as well as for managing and monitoring distribution network state. It contains integrated DMS analytical function system.

- SCADA system user interfaces.

2 Related Work

Whittaker and Thomason described a method for statistical testing based on Markov chain model of software usage in [14]. In this fundamental paper, authors discuss the construction of Markov chain and show how analytical results associated with Markov chains can aid in test planning. An innovation in this method is that test sequences generated and applied to the software are used to create the second Markov chain to encapsulate the history of the test. The paper also presents a stopping criterion for testing process. However, the example Markov chain for a hypothetical graphical user interface (GUI) is a very simple one. In the conclusions and prospects for future work authors mention that they are investigating more abstract models. To that end, we are still missing more complex and abstract domain specific models in the available literature. That is exactly the place where our paper tries to provide contribution by dealing with the construction of real-world operational profiles for the complex library of power distribution functions. The parameters of application programming interface (API) functions of this library are complex data structures rather than simple keystrokes. In contrast to [14], the stopping criterion in our paper is simply reaching the given target reliability figure.

In their essay on application of statistical science to testing and evaluating software intensive systems [5], Poore and Trammell present a high-level overview of the statistical testing process. In the section on building usage models they provide valuable rules and hints, e.g., how to expand states and arcs and how to create sub-models within a model, but they do not cover sensitive issue of how to generate inputs. Generating inputs gets really complicated when they are not simple types, such as strings, etc. For example, in the case of the DMS library the input may be the part, or complete model of the power distribution network, which is essentially a graph whose nodes and arcs are complex data types. In our paper, we provide some guidance and experience how to generate such inputs. Authors of [5] also present analytical method of assigning probabilities to state transitions within usage model. Alternatively, in our paper, we practice direct assignment of probabilities based on domain specific knowledge of power distribution systems, because they provide rather accurate representation of the reality. But, it is worth mentioning that analytic assignment has its place in practice when the probabilities are unknown.

Guen, Marie, and Thelin have proposed coverage measures for both states and transitions of the usage model and an approach to estimate the reliability from Markov chains by using their tool MaTeLo in [3]. After necessary definitions they introduce a heuristic for the construction of: equivalence classes, estimators in spaces of internal states and input vectors, and global indicators for individual transitions and for the whole chain. Then they present limitations of solutions proposed by Whittaker and Sayre, and propose their own method which combines the Sayre's solution and calculation based on the equivalence classes. In contrast to

[3], we do not measure model coverage directly. We use an alternative approach by selecting the test case length based on the domain-specific knowledge and then use significance quality indicators (SLi)s as the measure of generated test cases quality. The stopping criterion in this paper is that the mean significance confidence level is above 20% and that the target reliability figure is reached. Another difference between [3] and this paper is that we use the reliability estimation method proposed by Woit [15], which is based on the purely statistical hypothesis approach and therefore does not require construction of equivalence classes.

In another paper [4], closely related to [3], Guen and Thelin report on their practical experiences with statistical usage testing by means of their tool MaTeLo. The company's Alitec experience shows that the model construction takes between 0.5 and 4.5 days per KLOC (1000 lines of code). They have found that even for small software products, the size of the model would be very large, and have commented that it is probably the major drawback of that sort of modelling. In addition, they reported that time to finish testing was between 1.5 and 8.3 days/KLOC for 5 example projects, but they do not provide data about the target reliability and coverage. In our own experience, the most consuming part of modelling was gathering domain-specific knowledge of the behaviour of the power distribution network, e.g. knowledge about how switches and tap changers are operated. That activity lasted several months, and it is still work in progress. Once the basic knowledge was collected, we were able to construct various operational profiles within a working week each, so that time for a model per MLOC (million lines of code) became extremely low, e.g. 5/2x106 = 2.5 days/MLOC. Typical testing campaign in our experience lasts for 2-3 weeks, yielding time to finish test in the range of 7-11 days/MLOC.

MaTeLo [4] contains a usage model editor. Based on entered usage model, the back end test case generator generates automatically TTCN-3 test cases. Similarly, in [10], [9], [7], there is usage model editor based on GME. Depending on entered usage model, textual test cases are generated and executed in a JUnit based test bed. In both approaches, there are three separated steps:

1. Input of the model
2. Generation of test case
3. Execution of test case in test bed

Also in the prototype tool presented in [12], the test case generator and the test oracle are separated. In their approach, test case generator is Java framework used for testing Java classes. In this paper, we present an approach where steps 2 and 3 are joined. The first step is realized by writing C++ code.

The flat and hierarchical models supported by current statistical testing environments, such as MaTeLo [3], tend to become enormous very quickly as systems grow in complexity. This fact is recognized as major drawback in [4]. In his study [13], Weber proposes a solution to this issue. The proposed solution is to use parallel models because their usage provides exponential reduction in model size. Essentially, Weber creates an operational profile of a system by extending its requirement specification model. The approach is demonstrated with the example of Flight Guidance System (FGS) provided by Rockwell Collins. Although [13] presents an interesting approach, it requires a specification model in language such as RMSL-e (based on Requirements State Machine Language, RSML) as its input which may not be readily available in case of legacy systems and development of such models would be very costly. For example, the library we are dealing with in our paper contains millions of lines of FORTRAN code. Additionally, this legacy code is sequential by its very nature. Therefore, reducing the size of operational profile based on introduced parallelism is not possible.

Most of research on operational profiles is focused on operations and little is said about operation parameters. Shukla et al. [12] offer a solution to this issue by introducing support for defining constraints on individual parameters, relationships between different input parameters (of the same or different operation calls), and between output parameters of calls and input parameters of subsequent calls. Unfortunately, they do not report in details what kinds of constraints and relationships are supported and how. Although their work seems promising, they demonstrated it only on a three rather small examples, namely Stack, Symbol Table and Forest (of abstract syntax trees) with 35, 128, and 234 lines of code, respectively. In our paper, we deal with the complex legacy library of power distribution functions. We provide parameter constraints and relationships by translating them into C++ code that directly manipulates power distribution network model used by the library.

3 Implementation under Test

This DMS analytical function system is a component that contains the domain specific knowledge of DMS software. It implements comprehensive set of sophisticated algorithms that enable efficient design, optimal operation and decision making referring to the whole

equipment installed in the distribution network [6]. It realizes all technical tasks in distribution utilities in the following four modes of operation:

- Operation Management,
- Operation Planning,
- Development Planning,
- Simulation, Analysis and Training.

All analytical functions are developed on the basis of algorithms specially aimed for distribution networks, which enable performing both analysis and optimization of operation and development of very large radial and weakly meshed distribution networks.

The DMS analytical function system is implemented as a dynamic link library, named dmsapp.dll. This paper describes the method we have applied in the black-box testing of the library. The library is written in the FORTRAN programming language (version 10, as of today it is being ported to version 11). It is developed by development team, and tested by quality assurance team. It is an important module of the DMS system and realizes calculation of load flow, fault calculation, thermal monitoring and several other functions. Application Programming Interface (API) of the library contains functions for creation and destruction of the network model and invocation of specific calculations. There is a C language API for use in C applications. API functions will not be described here in detail. All calculations that the library and its API provide to developers are based on the software model of distribution network and this model is implemented in the library.

The software model of the distribution network in DMS system is a complex data structure. As has already been stated, it is the graph whose nodes and arcs are complex data types. Elements in the model are linked by identifications: for each element, in the data structure that represents it, there are fields for identifications of adjacent elements. In the general case, each network element is presented with three objects, containing different types of data. Those three objects comprise an abstraction hierarchy. The catalogue object contains catalogue data. Those are technical and administrative data that describe the class of objects. The set of similar objects after a period of use are assigned type data that is the result of statistical processing of measured data about the set. The basic data describe the specific element in case, and among others, contain the dynamic values. Thus, the catalogue object is the highest one in abstraction hierarchy and the basic data object is the lowest.

4 Testing Procedure

Testing of the library is realized using test environment we have developed. It is based on CPPUnit [1] framework for automated testing. In the setUp() method of CPPUnit test case, the test environment is prepared for the test. The tested library is initialized and the network model is loaded into the library. In the main test case method, which is registered to the CPPUnit framework, the finite state machine of Markov chain is executed. This is done by using OP State Machine class which is configured with particular states and their transitions. Each state transition implies invocation of functions that belong to the application programming interface of the tested library. In the tearDown() method, the test environment is closed down by freeing the used memory and by closing the handles of used system objects. It is possible that CPPUnit has defects, but during the testing we did not encounter them. If those defects manifested themselves, it would not make a problem for the testing process, because in that case, the asserts would fail.

The applied methodology is based on statistical testing methods described in [15], which are often used today, for example in Cleanroom Engineering [11]. This method implies that the functional correctness of DMS System is tested using operational profiles. The method is sufficient for testing provided that the operational profile correctly models the statistics of the usage of the tested product. An operational profile is modelled as a finite state machine. It can be represented as a graph, consisting of a certain number of states which are nodes of the graph. We can describe the state as a general condition of the software module. The edges are state transitions which are triggered by events that occur with certain probabilities. An event is either an external invocation of a member function of the tested module or a change of value of an externally accessible variable.

We have developed realistic, non trivial operational profiles for the DMS system. The API that is used for program invocation of DLL functions is rather complex because data structures that are used for data passing - as input and output arguments of function calls - are large and complex. This implies a large number of test cases required for testing of the analytical functions library.

The primary task of the testing tool is to generate requested number of test cases with the requested number of test steps. A test step represents an event that is defined in operation profile. The length of the test case represents the number of test steps in the test case. The more test cases are executed successfully, the greater

is the system reliability. An important characteristic of our approach is that tests are generated and executed dynamically. Test cases are generated (based on specified operational profile) during execution of the system in the test bed. This feature is realized by joining together the test case generator and the test bed.

In the traditional statistical testing, the first step is to generate (based on the operational profile) test cases and to save generated test case in a file. In the next step, the file is read and the test case is executed in the test bed. On the other hand, in our approach, the two steps are joined. First, the test bed generates the next test step by selecting one of the state transitions available in the currently active state - according to probabilities of state transitions and then, generated test step is executed. This is repeated until the last test step in the test case is executed. Operational reliability of the software module is the probability that module execution, selected at random according to given operational profile, will not fail.

Reliability = 1 - (failure rate).

Module execution is considered to be the sequence of events issued to the module beginning with module initialization (or re-initialization) and ending with module termination (or re-initialization). Operational profile is a description of the distribution of input events that is expected to occur in module operation. System reliability is determined by formula:

$M = r^N$

where:

N - number of successfully executed tests,

r - reliability,

M - confidence

For example, for the system's reliability (with the confidence of 0.7%) to be 99%, 500 test cases ought to be executed successfully, for the reliability of 99.9%, 5000 test cases ought to be executed successfully, for the reliability of 99.99%, 50000 test cases ought to be executed successfully, etc. Confidence is the probability that module has reliability less than r and still passes N tests. Significance level represents the quality of test sample of operation profile. Significance level is calculated using next formulas:

$e_{\mathrm{ij}} = P_{\mathrm{ij}} * \sum f_{\mathrm{ij}}$,

$D_{\mathrm{i}} = \sum (f_{\mathrm{ij}} - e_{\mathrm{ij}}) * \frac{f_{\mathrm{ij}} - e_{\mathrm{ij}}}{e_{\mathrm{ij}}}$,

where:

P_{ij} - probability of event j in state i,

f_{ij} - real occurrence of event j in state i,

e_{ij} - expected occurrence of event j in state i,

D_{i} - discrepancy of state i.

For significance level calculation, it is necessary to form a table in which rows represent states and columns represent events. Elements of the table are real and expected occurrence of events (eij and Pij). Significance level is determined in χ-table for every state based on calculated discrepancy. The mean value of the significance level is calculated next. It is given as the arithmetical mean value of significance levels of all states. If the mean value of significance level is greater than 20%, the given sample of operation profile is valid.

The testing method also includes a method for estimating software reliability based on statistical hypothesis testing. The result is reliability estimation accompanied by the measure of confidence. The overall procedure comprises of the following steps:

1. Specification of an operational profile
2. Generation of random test cases based on the operational profile
3. Execution of test cases
4. Verification of test cases
5. Reliability estimation based on the model and the results of the verification of test cases

Testing process is modelled using Binomial distribution because it satisfies the following criteria:

- Testing is performed with replacement
- Tests are selected at random and they are mutually independent
- Test has only two possible outcomes (success, failure)
- The probability of failure does not change during the testing

In this testing, three operation profiles have been identified and implemented:

1. Operational profile for changing dynamic data of distribution scheme. This profile randomly chooses a number of elements of each type and randomly changes dynamic values of selected elements. As each switchgear element contains purpose field, the elements are classified according to the value of that field. The set contains 5% of all switchgear elements with the purpose field indicating supply line or reactor, 10% of those with purpose indicating three types of high voltage transformers or a feeder, etc. After each change, load flow is calculated once again. The validity of Kirchhoff laws is checked before and after series of changes of dynamic values of different elements.

2. Operational profile for scheme mutation. This profile is different from profile 1 because instead of dynamic values, the static structure of the network is changed. There are two basic types of mutation: scheme extension and scheme reduction. Extension is achieved by adding elements or groups of elements, and reduction by deleting elements or groups of elements from the scheme.

3. "Realistic" operational profiles. We have developed six operational profiles based on operations that network operators routinely undertake in their everyday work. Those are:

- Change of the position of tap changers,
- Activation of capacitor batteries and its impact on currents and voltages,
- Load growth simulation and monitoring of changes of currents in the transformer bay,
- Detection location and insulation of faults,
- Feeder looping,
- Load sharing between the two feeders (Feeder to feeder load shedding).

We describe here a typical test case for operational profile 1. First, the tested library is initialized and an existing network scheme is loaded. This is a scheme that has already been used with the DMS system and thus, it is assumed to be correct. This starting statement is necessary because in further steps, the network scheme would be modified.

The test procedure modifies the scheme and checks if the library would successfully process or recognize as invalid the subsequent states of the scheme. The scheme is stored in a set of binary files and placed in the common folder. It is loaded from binary files to the library by first placing it into an array of binary type data structures. If this step goes well, the scheme is loaded from the array into dmsapp.dll.

Using one of the described profiles, the loaded network scheme is modified (by modifying dynamic values of network elements), see 1. By invoking the load flow calculation or by checking the validity of Kirchhoff laws, test case verifies if the DMS system can process the modified network. The expected behaviour is that the library should either correctly perform the requested calculations or return an error code in the case of an invalid state of the input network. Any other behaviour (e.g., unsuccessful calculation, throwing an exception during calculation) is considered to be test failure.

For the operational profile 2 (scheme mutation), the scheme is modified (by modifying the network structure) before it is loaded into dmsapp.dll.

From the programmer's point of view, test execution is achieved by instantiating an object of class OPStateMachine that models the finite state machine of module execution. First, objects that represent specific states of operation profile are instantiated and their state transitions (including probabilities) are configured. Thus a vector of state objects is formed. This vector is a parameter of the OPStateMachine constructor. During the construction of the object, the state machine is executed. By applying probabilities that are given in the state machine, the test case is generated and executed dynamically. Next the significance level of the test is calculated. This calculation is based on two matrices: of expected and achieved occurrence of events. Based on the calculated value, it is decided whether generated test case is a representative sample.

5 Operation Profile for Changing Dynamic Data

The operation profile for changing dynamic data has the following three states: S_0 - Initial state, S_1 - Network loaded, ready for check procedures, S_2 - Check procedure done, network ready for the next modification. and seven state transitions which are represented with events that trigger them and state transition probabilities we used in testing:

1. LoadDMIAndSetLibOptions (100%),
2. CheckKirchhoffsLaws (100%),
3. ChangeCapacitorChangers (15%),
4. ChangeFuseStatus (20%),
5. ChangeSwgStatus (35%),
6. ChangeTRTapChangers (15%),
7. SetDynValueGenerator (15%).

In the following text, each of the events is explained. The LoadDMIAndSetLibOptions event triggers scheme loading. The following options are set for dmsapp.dll: Load Flow, Estimation, Performance and Switching. All Switching options are activated except grounded and energized and different levels.

The CheckKirchhoffsLaws event triggers the procedure in which the validity of Kirchhoffs Laws for the network is checked. The ChangeCapacitorChangers event starts the procedure in which the list of all capacitors with variable capacity is compiled, and then

the capacity of each capacitor in the list is changed to a random value in acceptable interval. This interval is read in the catalogue object of the capacitor.

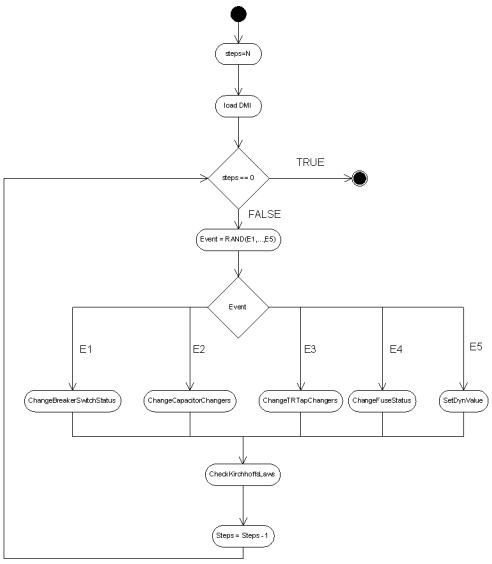

Figure 1: The UML activity diagram for the profile 1 (changing dynamic data)

The ChangeFuseStatus event starts the procedure in which the list of all fuses in the network is compiled. 10% of elements in the list are chosen at random and turned off, the rest are turned on. Before each turn on/off operation, the viability of the operation is checked, and upon each successful state change, it is checked whether the load flow calculation of the new network topology is possible. If not, the test is finished at that point.

The ChangeSwgStatus event starts the procedure in which the list of all switchgear elements in the network is compiled. Next, the elements in the list are classified by the value in their purpose field. For each purpose type, a set of randomly chosen elements A is formed. Set B, is subset of A and elements of B are chosen at random from A. The status of elements in B is set to open, and the status of all elements in A that are not in B is set to close.

The ChangeTRTapChangers event starts the procedure in which the list of all tap changers for high voltage transformers in the network is compiled. Next, values of elements in the list are randomly set in ranges that are read from their catalogue objects. This procedure is repeated for middle voltage transformers, too.

The SetDynValueGenerator event starts the procedure in which the list of all generators in the network is compiled. The regulation type of the generator can be:

1. Regulation of active power,
2. Regulation of active power and voltage,
3. Regulation of active and reactive power,
4. Regulation of active and reactive power and voltage.

Based on the regulation type, the values of elements in the list are changed:

1. Active power in the range (P_{min}, S_{nom}), where P_{min} is the minimal power of the generator and S_{nom} is the nominal power of the generator,
2. Reactive power in the range (Q_{min}, Q_{max}), where Q_{min} is the minimal reactive power and Q_{max} is the maximal reactive power
3. Voltage in the range (0.5*V_{nom}, 1.5*V_{nom}), where V_{nom} is the nominal voltage.

P_{min}, S_{nom}, Q_{min}, Q_{max}, and V_{nom} values are retrieved from the catalogue object for the generator. The probabilities of events are determined in the following manner. LoadDMIAndSetLibOptions and CheckKirchhoffsLaws have 100% probabilities to occur. The events ChangeTRTapChangers, ChangeCapacitorChangers and SetDynValueGenerator are assumed to be less probable to cause errors, so they have smaller probabilities. The events ChangeSwgStatus and ChangeFuseStatus are changing the topology of the network. Therefore, it is assumed that they can cause more errors and thus have greater probabilities.

Figure 2) presents the operational profile for changing dynamic data. There are three states. Upon loading the scheme, the system transitions from S_0 to S_1, and afterwards, upon checking Kirchhoff laws, it transitions from S_1 to S_2. It returns from S_2 to S_1 by changing dynamic values of certain network elements.

6 Feeder-to-Feeder Load Shedding

This operational profile is based on one of the scenarios of the realistic use of the DMS system. In the following paragraph the reader can find explanation of the testing procedure in this operational profile. The first step is to compile a list of all candidates. The candidate has to meet the following conditions:

- Switchgear element SWG is in off state
- SWG is contained in feeder bay TSM (transformer station medium voltage) or joint
- Bay should be connected to an energized bar

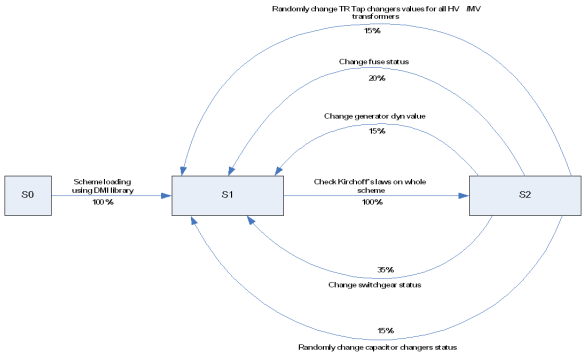

Figure 2: The operational profile for changing dynamic data

- SWG should be connected to two different feeders, belonging to different TSH (transformer station high voltage) elements

In the second step, an element from the list is selected at random (SWG_1) and the testing is started. Measured values for both feeders are recorded. The following variables are set: the feeder with greater load is F_{max}; TSH that contains F_{max} is TSH_{max}. Next, the direct path from TSH_{max} to SWG_1 over F_{max} is searched. A list of elements containing sections and TSMs in the direct path from TSHmax to the selected TSM is compiled. A list of all SWG elements that belong both to that path and to the feeder is compiled. If there are no SWG elements in the path that meet the requirements, the test is considered to be successful, and the execution returns to the beginning of step 2.

Next, SWG_1 is switched off. If it is not possible to execute load flow calculation, SWG_1 is switched to its former state, the test is considered to be successful, and the execution returns to the step 2.

From the list of SWG elements that are in the path and are switched on, one is selected (SWG_2). SWG_2 is switched off. If it is not possible to execute load flow calculation, SWG_1 and SWG_2 are switched back to their previous states, the test is considered to be successful, and the execution returns to the step 2.

Check of the validity of Kirchhoffs laws for the new state of the network is conducted. The new measured values for selected feeders are compared with old, and if the following conditions are met, the test is considered to be successful:

$Feeder1Load < Feeder2Load$:
$Feeder1Load < Feeder1LoadNew$
$Feeder2LoadNew < Feeder2Load$
$Feeder1Load > Feeder2Load$:
$Feeder1Load > Feeder1LoadNew$
$Feeder2LoadNew > Feeder2Load$

Afterwards, SWG_1 and SWG_2 are switched back to their previous states and the check of the validity of Kirchhoffs laws for the new state of the network is conducted. The execution returns to the beginning of the step 2.

The test is performed in the following manner. First, from the list of SWG_1, one by one, each of the elements in the list is selected. Next, the path from TSH to TSM containing SWG_1 is searched (over the feeder with greater load). Afterwards, all SWG elements in the path are added to the SWG_2 list and a SWG is selected at random from the SWG_2 list. In the last step, feeder to feeder load shedding is performed for the selected combination.

7 Conclusion

DMS is a large and complex software package for performing technical tasks in electric power distribution utilities. As already stated, DMS software systems are getting more complex every day. One important software metric is its reliability, and therefore, statistical testing by using operational profiles as a de facto standard for measuring software reliability is of special interest in development of DMS systems.

Two important contributions of this paper are the set of developed operational profiles for statistical testing of DMS systems and on-the-fly testing approach, which is achieved by joining together the test case generator and the test bed. In earlier test beds that are described in the available literature, the two were separated.

The operational profiles have been developed for testing of one module of distribution management system software - that is the module that contains analytical functions subsystem. Several operational profiles have been developed, two of which are presented here in detail (operational profile for changing of dynamic data, and feeder-to-feeder load shedding).

Besides providing us with a measure of reliability, the testing with the use of operational profiles has allowed us to identify certain software faults earlier in the development process.

8 Acknowledgement

This work was partially supported by the Ministry of Education and Science of the Republic of Serbia under the project No. 44009 and 32031, year 2011.

Authors thank Pavle Kuzevski, Jelena Djurica, Tereza Kovac, Snezana Crnogorac Jovanovic, Ranka Slijepcevic, Rodoljub Radivojevic and Ljiljana Dragojlic for implementation of operation profiles, and valu-

able discussions and feedback during realization of this paper.

References

[1] Cppunit - c++ port of junit. sourceforge.net/projects/cppunit.

[2] Dms software, windows for distribution networks. DMS Group, 2006.

[3] Guen, H. L., Marie, R., and Thelin, T. Reliability estimation for statistical usage testing using markov chains. In *International Symposium on Software Reliability Engineering (ISSRE)*, 2004.

[4] Guen, H. L. and Thelin, T. Practical experiences with statistical usage testing. In *Annual International Workshop on Software Technology and Engineering Practice (STEP)*, 2004.

[5] Poore, J. H. and Trammell, C. J. *Statistics, Testing, and Defense Acquisition*, chapter Application of Statistical Science to Testing and Evaluating Software Intensive Systems. National Academy Press, 1998.

[6] Popovic, D., Bekut, D., and Treskanica, V. *Specijalizovani DMS algoritmi*. DMS Group, 2004.

[7] Popovic, M., Basicevic, I., Velikic, I., and Tatic, J. Practical experiences with statistical usage testing. In *Annual Conference on Engineering of Computer Based Systems (ECBS)*, pages 377–386, 2006.

[8] Popovic, M., Basicevic, I., and Vrtunski, V. Practical experiences with statistical usage testing. In *Annual Conference on Engineering of Computer Based Systems (ECBS)*, 2009.

[9] Popovic, M. and Kovacevic, J. A statistical approach to model-based robustness testing. In *Annual Conference on Engineering of Computer Based Systems (ECBS)*, pages 485–494, 2007.

[10] Popovic, M. and Velikic, I. A generic model-based test case generator. In *Annual Conference on Engineering of Computer Based Systems (ECBS)*, pages 221–228, 2005.

[11] Prowell, S. J., Trammell, C. J., Linger, R. C., and Poore, J. H. *Cleanroom Software Engineering: Technology and Process*. Addison-Wesley Professional, 1999.

[12] Shukla, R., Strooper, P. A., and Carrington, D. A. Tool support for statistical testing of software components. In *Asia-Pacific Software Engineering Conference (APSEC)*, 2005.

[13] Weber, R. J. Statistical software testing with parallel modeling: A case study. In *International Symposium on Software Reliability Engineering (ISSRE)*, 2004.

[14] Whittaker, J. A. and Thomason, M. G. A markov chain for statistical software testing. *IEEE Transactions on Software Engineering*, 20(10), October 1994.

[15] Woit, D. M. *Operational Profile Specification, Test Case Generator, and Reliability Estimation for Modules*. PhD thesis, Queen's University Kingston, Ontario, Canada, 1994.

Applying the Heterogeneity Level Metric in a Distributed Platform

PAULO S. L. SOUZA
FABIO HISTOSHI
MARCOS J. SANTANA
REGINA H. C. SANTANA
SARITA M. BRUSCHI
KALINKA R. L. J. C. BRANCO

USP - University of São Paulo
ICMC - Institute of Mathematics and Computer Sciences
SSC - Computer Systems Department
P.O. Box: 668 - 13560-970 - São Carlos (SP) - Brazil
{pssouza, hitoshi, mjs, rcs, sarita, kalinka}@icmc.usp.br

Abstract. Heterogeneity Level (HL) metric has been developed by our research-group to help scheduling algorithms to adapt themselves to the existent heterogeneity in the platforms. This paper presents our results considering the HL's behaviour in a real adaptive scheduling. HL metric quantifies qualitative aspects from heterogeneity in order to provide efficient performances and lower cost to the execution in both heterogeneous and homogeneous platforms. HL use is investigated under different perspectives: CPU, memory, network and considering benchmarks results. A simple but effective adaptive scheduling using HL is proposed and its results point out to performance-gains around 53% when a non-adaptive scheduling algorithm is used. Our case studies show that the HL was efficient, flexible and easily used for scheduling policies.

Keywords: heterogeneity, load balancing, cluster.

(Received February 22nd, 2011 / Accepted May 2nd, 2011)

1 Introduction

Heterogeneous distributed platforms allow exploring different and specific resources according to different demands. They extend the platform performance through both gradual improvements and reuse of the resources already available in the organization. However, associating different resources with diversified demands implies to compute a more complex strategy for this distribution. Heterogeneity must be used carefully in order to improve the computing cost vs. benefit relation.

Processes scheduling is directly affected by heterogeneity. It is necessary to consider relevant aspects from both platform and applications demand when the resources present different features, architecture or performance. On the other hand, when the platform is homogeneous, the scheduling may encapsulate details from devices and basic software (such as operating systems and compilers), because they present a uniform behavior, performance and architecture. This allows simpler and cheaper scheduling with efficiency. Other important point is that heterogeneity can be temporal as well, due to workload dynamical variation and node-changes in the platform [3].

Branco et al. [3] proposed the HL (Heterogeneity Level) metric to quantify the platform heterogeneity, considering the performance's dispersion from each node, in relation to an average performance [3]. Their preliminary results show the HL performance under simulation. This paper presents our main results from considering the HL's metric in a real adaptive schedul-

ing for distributed platforms. Our aim in this paper is compare the HL metric behavior when applied in an adaptive scheduling policy on a platform with different heterogeneity levels.

The HL behavior is investigated using two different perspectives: changes in the hardware-resources and with distinct benchmarks. These hardware resources and benchmarks allow analyzing the HL behavior according to different and real perspectives, such as: floating-point, integer operations, memory use and network consumption [6, 7, 10, 11, 12, 14, 15].

A novel adaptive scheduling algorithm has been proposed to investigate HL impact in this context. This algorithm is used by AMIGO (DynAMical FlexIble SchedulinG EnvirOnment) [13] and PVM (Parallel Virtual Machine) [5]. The choice by PVM is due to both its source code structure and its tightly coupling to AMIGO. However, the PVM choice does not imply in generality loss, because the studies presented here are focused on the HL impact mainly, independently if this scheduling is done by MPI, PVM or by a distributed operating system. Indeed, the novel scheduling policy and the investigations about the use of the HL are orthogonal to the message passing interface used.

The best results in our case studies show that the adaptive scheduling using HL metric allow a real performance gain around 53% for the application runtime. The HL is simple to be used in scheduling algorithms and presents a stable behaviour.

This paper is organized as follows. Section 2 presents some basic definitions about heterogeneity. Section 3 presents HL metric and its behavior considering benchmarks demand. Section 4 presents the adaptive scheduling algorithm proposed, describing how to use the HL metric. Section 5 describes the main results obtained using the HL metric and Section 6 presents the concluding remarks.

2 Heterogeneity and Homogeneity

There are different kinds of heterogeneity [3]. Initially, it can occur considering the configuration and architecture. There is configuration heterogeneity when differences of performance are observed on devices with the same platform (hardware and basic software). Architectural heterogeneity implies different devices when considering hardware and/or basic systems.

Heterogeneity can be considered as positive or negative, depending on heterogeneity contribution for the system performance. We have a positive heterogeneity when devices with better performance in relation to previous ones are added in the platform. A negative heterogeneity (or a performance lack) can occur when devices with worse performance are added in a platform [3].

Time is other important feature related to heterogeneity. There is temporal or dynamical heterogeneity if the platform presents a homogeneous behavior in determined situations and heterogeneous in other. Factors that contribute for a temporal heterogeneity are: workload, multi-users and the resources being considered to report the heterogeneity level. Considering the last one (resources), for example, when two nodes have both distinct processors and the same memory quantity/type, they can be heterogeneous under CPU point-of-view and homogeneous when considering memory. Depending on application demand, the platform can be considered heterogeneous or homogeneous [3].

Different research works quantify the platforms heterogeneity degree [1, 3] [6, 14, 16]. Some models and metrics for heterogeneous systems were proposed by Zhang and Yang [16], in which heterogeneous computing systems can be represented by a graph (M, C), where $M = M_1, M_2, M_3, M_4, M_5, ..., M_n$ is considered a set of heterogeneous workstations and C is the communication network linking the workstations (with a homogeneous bandwidth). Aiming to quantify the heterogeneity of a system machines without using complex measurements, Zhang and Yang [16] proposed two metrics to evaluate the relative computing power of a set of workstations (the capacity of each workstation is evaluated in comparison to the fastest one):

$$W_i(A) = \frac{S_i(A)}{max_{i=1}^{n}\{S_i(A)\}} \quad (1)$$

Where $i = 1, ..., n$ and $S_i(A)$ represents the speed of M_i to execute application A dedicatedly. Speed can be defined by the number of basic operations per time unit, for instance, and the computing power of each workstation is represented by a relative speed. A second metrics proposed is:

$$W_i(A) = \frac{min_{i=1}^{n}\{T(A, M_i)\}}{T(A, M_i)} \quad (2)$$

Where $i = 1, ..., n$ and $T(A, M_i)$ is the time required to execute application A at workstation M_i. Grosu [6] extends these metrics so that the computing power is given by the relative speed of the workstation in relation to the slowest one:

$$W_i(A) = \frac{min_{i=1}^{n}\{S_i(A)\}}{S_i(A)} \quad (3)$$

Where $i = 1, ..., n$ and $S_i(A)$ is the speed of workstation M_i to execute application A dedicatedly, and the

computing power is given by relative speeds. Furthermore, Grosu [6] defines:

$$W_i(A) = \frac{T(A, M_i)}{max_{i=1}^{n} \{T(A, M_i)\}} \quad (4)$$

Where $i = 1, ..., n$ and $T(A, M_i)$ is the time it takes to execute application A at workstation M_i.

Thus, equations 1 and 2 now act as the basis to define the computing power, considering the fastest machine as a reference point, which is renamed $W_i^f(f - fast)$. On the other hand, equations 3 and 4 identify the computing power based on the slowest machine, which is represented by $W_i^s(s - slow)$. Four ways to quantify the heterogeneity level in a system based on the value of W are proposed in [16] and [6]. The first and second use the standard deviation H_1, which can be calculated based on the computing powers compared to either the fastest or the slowest workstation:

$$H_1 = \sqrt{\frac{\sum_{i=1}^{n}(W_{med} - W_i)^2}{n}} \quad (5)$$

The mean absolute deviation, called H_2, also calculated based on the fastest or the slowest workstation:

$$H_2 = \frac{\sum_{i=1}^{n} |W_{med} - W_i|}{n} \quad (6)$$

where:

$$W_{med} = \frac{\sum_{i=1}^{n} W_i}{n} \quad (7)$$

The values in both H_1 and H_2 are observed and analyzed uniformly, using the average to find the standard deviation and the mean absolute deviation. However, this uniformity invalidates the analysis when there are reasonable differences among the workstations computing powers, since the standard deviation cannot reflect computer systems.

Based on this weakness of the H_1 and H_2 metrics, Zhang and Yang [16] proposed a third metric, H_3, evaluated from the fastest workstation in the computing system:

$$H_3 = \frac{\sum_{i=1}^{n}(1 - W_i^f(A))}{n} \quad (8)$$

Similarly, Grosu [6] defines H_4 based on the computing power of the slowest workstation in the computing system:

$$H_4 = \frac{\sum_{i=1}^{n}(1 - W_i^s(A))}{n} \quad (9)$$

In H_3, the computing power of the fastest workstation is equal to 1 while, in H_4, the slowest machine has a computing power value of 1. Thus, H_4 represents the difference of computing power between each machine and the fastest machine and H_3 calculates the same difference between each machine and the slowest one.

Based on his experiments, Grosu [6] states that the metric H_4 is more suitable than H_3. However, the case studies presented in [3] demonstrate the fallacy of that statement in some situations, because the metrics present contradictory behaviour when evaluating the same platform, in different cases.

Branco et al. [3] propose the *HL* (Heterogeneity Level) metric to eliminate these discrepancies. *HL* considers a hypothetic (not real) standard node, representing the nodes average performance [3]. This metric will be detailed in next section due to its importance for this work.

3 The HL Metric

Platform heterogeneity can be quantified considering different perspectives, such as: architectural aspects, operating systems or resources performance [15]. In this sense, Branco et al. proposed the *HL* metric [3] to quantify the heterogeneity, using a virtual node (called standard node), which represents the average performance in the platform. The dispersion around this standard node allows quantifying the platform heterogeneity level, in a similar way to the works presented in [6] and [16] that use respectively the worst and the best node as standard node. The main difference between *HL* metric and those two is that *HL* has a uniform behaviour when quantifying distinct heterogeneity levels and also both the positive and the negative heterogeneities. *HL* quantifies the platform heterogeneity level through equation 10, where n represents the amount of nodes in the platform, X_i is the $node_i$ performance and $\overline{X}$ is the virtual standard node performance.

$$HL = \frac{\sum_{i=1}^{n} \left|X_i - \overline{X}\right|}{n * \overline{X}} \quad (10)$$

The preliminaries results using the *HL* [3] considered a general parameter called "speed", in order to qualify application demands. Indeed, Branco et al. considered "speed" as a generic value, which should be easily instantiated later at real parameters, such as: MIPS, MFLOPS, runtime, RAM amount or other.

Different experiments were conducted by Branco et al. [3] to simulate the *HL* behaviour when including new nodes in a heterogeneous platform with just three nodes with "speeds" 10, 100 and 1000.

Figure 1 show the *HL* behaviour when nodes identical to the fastest one are added. The heterogeneity degree behavior is coherent, since as similar high-speed

nodes are being added, the system heterogeneity level drops and stabilizes close to zero. Few nodes with high-speeds change the platform status quickly for homogeneous. This situation could be represented by the metrics proposed by Zhang (Zhang & Yang) as well; however, it is not properly represented by Grosu's metrics [6], due to standard node, based on the slowest workstation.

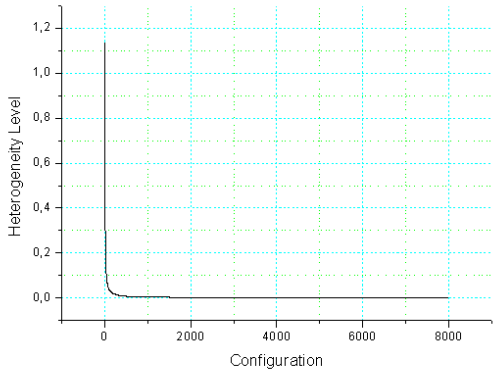

Figure 1: Behavior of the heterogeneity degree when nodes identical to the system's fastest node are added (initial speeds of 10, 100 and 1000)

Figure 2 shows the *HL* behaviour when nodes identical to the slowest one are added. The heterogeneity level rises at a first moment, because more nodes with "speed" 10 are required to reach the speed of the fastest node. When they reach this threshold, the heterogeneity level falls near to zero, where it must stabilize. The *HL* metric evaluations done by Branco et al. [3] were not concerned about static or dynamic behaviour of the environment being used for the experiments, since the "speed" parameter encapsulated this question.

However, metrics based on static data, such as hardware features, offer just a partial view of both performance and heterogeneity. They are not able to represent usual dynamic changes in the platforms that make them temporarily heterogeneous. In this sense, dynamic metrics, such as runtime, can point out the heterogeneity level on-the-fly and according to user point-of-view [12]. Runtime encapsulates basic details from hardware and software, grouping them in a common point: to offer better performance (considering time) to end-user applications. Indeed, if used properly, time allows a performance comparison while encapsulating architecture details.

We develop initially two new experiments with *HL* metric to show these perspectives. For the first one

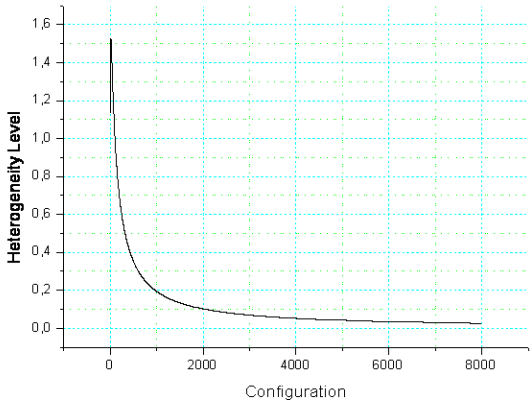

Figure 2: Behavior of the heterogeneity degree when nodes identical to the system's slowest node are added (initial speeds of 10, 100 and 1000)

we use real and static hardware features to evaluate the *HL* metric. For the second one we use dynamic benchmarks results to determine the heterogeneity. Both experiments consider the same platform. The hardware features considered for the first analysis are: CPU frequency, cache amount, RAM amount, swap-memory amount and network peak throughput (see Table 1). The second analysis considers six different open-source benchmarks: (Whetstone, Dhrystone and Linpack), memory (Stream and Cachebench) and network (Netperf). Table 2 points out the main features evaluated in each benchmark and their respective metrics [4, 7, 9, 10, 11, 14]. These benchmarks were chosen because they are: (1) meet the specific-demands planned for our experiments, (2) open-source and (3) free.

The experiments were executed on a cluster with 5 nodes, all using GNU/Linux, distribution OpenSuse 10.0, 100Mbits ethernet network and gcc compiler. Table 1 contains the *HL* resulting from each hardware feature analyzed, where it is possible to observe a stable *HL* behavior. This platform can be viewed as homogeneous if the network maximum throughput is considered and as heterogeneous one if the CPU performance is taken account. The demand generated by application should determine which metric must be used to reach effectiveness when using the heterogeneity level with these data. This implies in a previous study of the demand and usually is associated with monitoring software tools, which automates this process and helps managers to characterize the demand in a correct way [8].

Table 1: *HL* results when evaluated on a five nodes cluster and according to hardware features such as: CPU frequency, cache amount, RAM amount, swap amount and network peak throughput.

Features	*Node1*	*Node2*	*Node3*	*Node4*	*Node5*	*HL*
CPU (MHz)	400.91	451.05	1200.07	1666.73	2017.99	0.50
Cache (KB)	512	512	64	256	512	0.45
RAM (MB)	192	128	256	256	256	0.21
Swap (MB)	196	243	415	512	256	0.34
Network (Mbps)	100	100	100	100	100	0.00

It can be observed through Table 3 that *HL* metric is also efficient to represent the platform heterogeneity and it presents a stable behaviour according to benchmarks results. The values showed in the Table 3 represent the average of 30 executions.

The *HL* values in Tables 2 and 3 point out some possible discrepancies that occur when considering heterogeneity in a distributed platform. An example is the *HL* value obtained from CPU feature (0.50) and the Whetstone result (0.74), because both consider CPU performance. In these cases, the results obtained from the benchmark were considered more efficient to represent the platform heterogeneity, since they intend to indicate the real performance for the user.

This difference can also be observed with memory and network. Network is a critical case, since the platform is homogeneous (*HL*=0.00) when considering peak performance (100Mbps). However, Netperf benchmark shows that when different nodes send messages to (or receive from) node 5, the platform presented the second major *HL* result (0.67) and thus, can be considered heterogeneous. Again, in these cases, the benchmark results should be used because they represent the performance expected by final user. In these cases, a simple view considering just one hardware feature is not a better choice to estimate the heterogeneity level.

4 An Adaptive Scheduling *As/Hl*

An adaptive scheduling based on the heterogeneity level (or *As/Hl*) was developed in this work to investigate the *HL* metric impact in a real scenario, considering the end-user perspective. The *As/Hl* is adaptive because it changes dynamically the scheduling algorithm according to the *HL* metric.

AMIGO (DynAMical FlexIble SchedulinG EnvirOnment) [13] and PVM (Parallel Virtual Machine) [5] were used to insert the *HL* metric in the *As/Hl*. AMIGO allows grouping specific scheduling policies according to different demands. The choice by PVM is due to its source code structure and because it is tightly coupled to AMIGO. However, it is important to note that the choice by AMIGO/PVM does not cause generality loss, because this policy could be applied in other contexts, such as: MPI environment, operating system or directly inside parallel application code. The *As/Hl* determines which scheduling algorithm must be used considering the platform heterogeneity. The aim is minimizing scheduling costs and at the same time maximizing its benefits.

Table 2: Benchmarks used to evaluate the *HL* metric behaviour.

Category	*Benchmark*	*Demand*	*Metric*
CPU	Whetstone	Floating-Point	Execution Time
	Dhrystone	simple arithmetic, strings, logical and access to the memory	execution time and dhrystones; performance in relation to Vax 11/780 for one benchmark iter
	Linpack	linear eq systems float/double in arrays	FLOPS and execution time
Memory	Stream	memory throughput	throughput and avg execution time
	Cachebench	accesses to memory and to cache	throughput
Network	Netperf	latency, TCP/UDP throughput	throughput and avg execution time

Table 3: *HL* results when evaluated on a five nodes cluster and according to six distinct benchmarks.

Benchmark	*Node1*	*Node2*	*Node3*	*Node4*	*Node5*	*HL*
Whetstone (s)	1119.0	995.4	265.0	190.4	1265.0	0.74
Dhrystone (s)	3.2	2.8	1.0	0.7	0.8	0.49
Linpack (s)	0.020	0.018	0.006	0.004	0.005	0.52
Stream (s)	0.31	0.30	0.07	0.06	0.04	0.58
Cachebench (Mb/s)	984.2	1105.0	3509.6	5054.5	5400.3	0.53
Netperf (Mb/s)	64.9	5753.3	5559.5	15988.2	x	0.67

AMIGO is basically composed by an upper and a

lower layer. Upper layer is responsible by the configuration, while lower layer is responsible by scheduling policies, AMIGOD (AMIGO Daemon), message-passing environment (in this work instantiated by PVM) and parallel applications [13].

AMIGO has scheduling policies for memory-bound, network-bound and CPU bound applications. DPWP (Dynamical Policy Without Preemption) is one of them, which presents features such as: dynamic (decides the scheduling at runtime), specific to CPU-bound applications and does not consider preemption [2]. The scheduling done by DPWP aims to balance new workloads considering the existent nodes load. It tries to normalize this workload using a relative-performance in relation to the whole platform. DPWP normalizes the ready-processes in the ready-queue to determine the target node.

AMIGO acts just when the message-passing environment needs to schedule new processes on the platform. Figure 3(a) shows the steps followed by the original PVM in this case. An application requests the scheduling from $pvm\_spawn()$, which forwards the request to local PVMD. The local PVMD, running $tm\_spawn()$, create a list contending nodes that will receive the new processes using by default a round-robin policy. The functions $assign\_tasks$ and dm_exec are called later to create and to start these processes, respectively.

To interact with AMIGO, $tm\_spawn()$ routine was modified (Figure 3(b)), where processes selection is always requested to AMIGOD through the $GetHostsFromAMIGOD()$. This request will be attended by an AMIGO's scheduling policy, independently if round-robin policy existing in PVM could present better results or not, according to used platform. The *HL* metric was inserted in this scenario trying to solve this problem by analyzing the platform heterogeneity and creating a simple but efficient adaptive scheduling algorithm in this point.

AMIGO is requested to give a nodes-relation according to DPWP policy when the platform is heterogeneous; when it is homogeneous, the request is not sent to AMIGO and the original round-robin policy is used to determine the nodes target to the scheduling. Figure 3(c) shows the algorithm basic steps, highlighting just the $tm\_spawn()$ routine. This adaptive algorithm is based on a conditional structure comparing the *HL* value returned from $eval\_HL()$ with a threshold, called $standard\_HL$. The correct choice of this value is a complex question and needs to be investigated in a more detailed sense. Unfortunately, this study does not belong to the scope of this paper. The $standard\_HL$ must be defined by the system manager considering the expected demand, used platform and objectives. The $standard\_HL$ was empirically fixed as 0.01 in this paper, as explained in the next section.

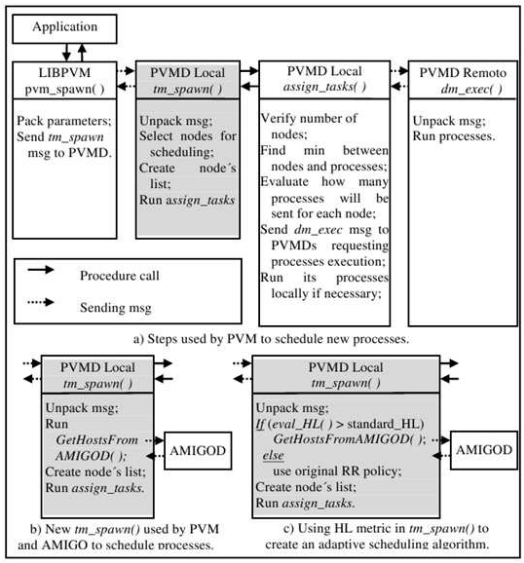

Figure 3: Steps followed by (a) original PVM, (b) PVM/AMIGO and (c) As/Hl algorithm in order to schedule new processes.

The *HL* is evaluated by the $eval\_HL()$ considering the nodes performance. The activities conducted by $eval\_HL()$ are: gets the computing performance from each node, evaluates the *HL* through equation 10 and returns the *HL* value.

The node performance can be determined from different ways. In this paper, they were instantiated through benchmarks previously executed.

5 Experiments and Results with *As/Hl*

The objective of the experiments with the *As/Hl* is to analyze the *HL* efficiency when it is applied in scheduling policies on distributed platforms. To reach this objective, it was developed an experimental study using a parallel application responsible to solve linear systems, based on Gauss-Jacobi iterative method. Gauss-Jacobi application was chosen because it is CPU-bound application, representative for a large number of HPC programs. The version executed in this work is compound by a master code responsible by dynamically generate slaves, these ones able to solve a variable sub-group from the linear system. The new slave processes generation is made on-the-fly by master processes. The master code evaluates the linear system convergence and, before starts a new iteration, it decides either to stop or

not the execution.

Experiments were done in a Beowulf cluster with ten nodes, all of them with: Intel Pentium4 Processor (64bits and 3.4GHz), RAM with 4GBytes, a Gigabit Ethernet network and GNU/Linux operating system.

This cluster is a homogeneous platform. Fixed-and-extra workloads were inserted into some nodes, turning this platform a temporally-heterogeneous one with three different levels: totally homogeneous, partially heterogeneous and totally heterogeneous. The synthetic extra workloads were generated using the Linpack, Dhrystone and Whetstone benchmarks. The homogeneous scenario is composed by ten nodes without any extra workload. In the partially heterogeneous scenario the nodes are grouped into five distinct pairs with workload ranging from 0% to 40% in relation to CPU utilization. For the totally heterogeneous scenario the nodes are also grouped in pairs, being generated for them extra workloads ranging from 0% to 80% in relation to the CPU utilization. Benchmarks used to create the two heterogeneous-scenarios affected the platform performance in a fixed and constant way, during the whole experiment.

The Linpack benchmark was used to evaluate the node performance in these three platforms. It was chosen because it has features close to the parallel application used: operations with floating-point vectors to solve linear systems. The results obtained from the Linpack were applied to equation 10 in order to evaluate the *HL* metric. Using Linpack to establish the heterogeneity in platform allows focusing on CPU features, majorly those related to FPU. Table 4 presents the results when using Linpack and *HL* for the three scenarios discussed.

Table 4: Results for Linpack benchmark and *HL* metric.

Scenarios Used						
	Totally Homogeneous $HL = 0.0002$		Partially Heterogeneous $HL = 0.22$		Totally Heterogeneous $HL = 0.57$	
Nodes	Extra Load	Time (ms)	Extra Load	Time (ms)	Extra Load	Time (ms)
1	0%	0.506	0%	0.505	0%	0.505
2	0%	0.504	0%	0.504	0%	0.504
3	0%	0.505	20%	0.684	20%	0.684
4	0%	0.504	20%	0.684	20%	0.684
5	0%	0.505	40%	1.074	40%	1.073
6	0%	0.504	40%	1.072	40%	1.072
7	0%	0.505	10%	0.585	60%	1.965
8	0%	0.504	10%	0.585	60%	1.964
9	0%	0.505	30%	0.907	80%	4.944
10	0%	0.504	30%	0.907	80%	4.940

The threshold used by *HL* to determine if the platform is either homogeneous or heterogeneous ($standard\_HL$) was arbitrarily fixed in 0.1, due to *HL* results obtained from the three platforms. This *HL* value allows separating the totally homogeneous platform (*HL*=0.0002) from other two possibilities: partially and totally heterogeneous with *HL* 0.22 and 0.57, respectively.

In a first execution, the Gauss-Jacobi application was scheduled using the *As/Hl* algorithm on the three platforms and according $standard\_HL$ value. This means that the policy used was the round-robin when executing the homogeneous scenario and the DPWP when executing on the partially and totally heterogeneous platforms.

In order to compare the scheduling done for each platform, we repeated the executions, changing the policies used. In these complementary executions the policy DPWP was chosen when executing the homogeneous scenario and the round-robin policy was the option when executing both the partially and totally heterogeneous platforms.

Table 5 presents the runtime average in seconds for thirty Gauss-Jacobi parallel application executions, using the round-robin/PVM and the DPWP/AMIGO/PVM scheduling policies, and considering platforms: homogeneous, partially heterogeneous and totally heterogeneous. The values between parentheses indicate the complementary execution.

Table 5: Gauss-Jacobi parallel application runtime using round-robin and DPWP policies on three different platforms. Values between parentheses indicate the complementary executions to compare the correct scheduling done in each platform by *As/Hl*.

Schedule Policy	*Totally Homogeneous*	*Partially Heterogeneous*	*Totally Heterogeneous*
Round-Robin	5,1s	(17,6s)	(28,6s)
DPWP	(7,8s)	11,6s	23,7s

The round-robin policy in the homogeneous scenario presents a better performance when compared to the results from the DPWP. These results point-out a 53% performance loss, in this case. The DPWP spent more time to find target nodes to receive new processes, while the original round-robin policy distribute these same processes equally among the nodes, doing scheduling in a simple and efficient way. Since the platform used is homogeneous and there were not external interferences from other applications, concurring to the available resources, the DPWP is unnecessary and inefficient. In this case the *HL* metric is capable to prevent

the use of a higher computing cost scheduling policy (such as the DPWP).

The DPWP policy is more efficient than the round-robin policy when considering the partially heterogeneous platform. In this case the performance gain was around 51.7%. This is due to the heterogeneity presented in the platform, fact considered only by the DPWP. Again, the *HL* metric can choose properly the schedule policy.

The DPWP obtains a better performance when compared to the round-robin in the heterogeneous scenario, as expected. However, the difference between both executions was lower, with a DPWP performance gain just around 20.7%. This smaller difference, when comparing to partially heterogeneous platform, is due to the overload of 60% and 80% in four nodes available for the experiments. These four nodes are near to saturation and this causes a higher impact in the DPWP performance, due to its costs. In this scenario DPWP spent more time to find the correct nodes to use and how many processes each node should receive, when comparing to the round-robin policy.

6 Concluding Remarks

This work investigates the *HL* metric behaviour in real scenarios. The *HL* metric [3] was investigated considering static and dynamic perspectives and it was also used in the *As/Hl* algorithm. The *HL* metric presented excellent behaviors in our case studies, pointing out the platform heterogeneity for both static features (e.g.: CPU frequency, memory quantity or network) and dynamical features, these obtained from benchmarks.

The results obtained from the experiments conducted with *As/Hl* algorithm, show performance loss to 53% when using the wrong scheduling policy in relation to the platform heterogeneity level. They show also performance gains to 51.7% when using the correct scheduling policy. The investigations performed in this work confirm that the correct use of the heterogeneity level is essential to improve the platform performance, therefore, producing better benefits with lower costs for the end-user. They also show that the *HL* is efficient to represent the heterogeneity degree, flexible when considering different heterogeneity perspectives and easy to be used in the processes scheduling context.

Future works include studying the *HL* metric thresholds to indicate if a platform must be handled either as homogeneous or heterogeneous, also under different perspectives such as: CPU, memory, network and a mixing of them.

7 Acknowledgments

The authors would like to thank CAPES, CNPq and FAPESP, Brazilian funding agencies, for the financial support.

References

[1] Al-Jaroodi, J., Mohamed, N., Hong, J., and Swanson, D. Modeling parallel applications performance on heterogeneous systems. In *Int. Parallel and Distributed Processing Symposium*, pages 160.2–, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Araujo, A. P. F., Santana, M., Santana, R. H. C., and Souza, P. S. L. Dpwp - a new load balancing algorithm. In *5th Int. Conference on Information Systems Analysis and Synthesis - ISAS'99*, Orlando, U.S.A., 1999.

[3] Branco, K., Santana, M., and Santana, R. H. C. A novel metric for checking levels of heterogeneity in distributed computer systems. In *Advances in Intelligent System and Robotic*. IOS Press, 2003.

[4] Curnow, H. J. and Wichmann, B. A. A synthetic benchmark. *Computer Journal*, 19(1):43–49, 1976.

[5] Geist, G. A., Beguelin, A., Dongarra, J. J., Jiang, W., Manchek, R., and Sunderam, R. Pvm 3 users guide and reference manual. Oak National Lab., 1994.

[6] Grosu, D. Some performance metrics for heterogeneous distributed systems. In *Proceedings of PDPTA'96*. Las Vegas, 1996.

[7] Linpack. www.math.utah.edu/software/linpack.html#documentation, Last access: 02/02/2011.

[8] Massie, Matthew, L., Chun, Brent, N., and Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[9] McCalpin, J. D. Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, 1995.

[10] Mucci, P. J. and London, K. The cachebench report, 1998.

[11] Netperf. www.netperf.org/netperf/NetperfPage.html Last access: 02/02/2011.

[12] Petterson, D. A. *Computer organization and design: the hardware/software interface*. Elsevier/Morgan Kaufmann, third edition, 2005.

[13] Souza, P. S. L., Santana, M., and Santana, R. H. C. Amigo - a dynamical flexible scheduling environment. In *5th International Conference on Information Systems Analysis and Synthesis - ISAS'99*, 1999.

[14] Weicker, R. P. Dhrystone: a synthetic systems programming benchmark. *ACM Computing Surveys*, 27:1013 – 1030, 1984.

[15] Zhang, Z. and Seidel, S. Benchmark measurements of current upc platforms. In *19th IEEE International on Parallel and Distributed Processing Symposium*, 2005.

[16] Zhang, Z. and Yan, Y. Benchmark measurements of current upc platforms. In *7th IEEE Symposium on Parallel and Distributed Proceeding*, pages 25–34, 1995.

An Adaptive and Historical Approach to Optimize Data Access in Grid Computing Environments

RENATO PORFIRIO ISHII[1]
RODRIGO FERNANDES DE MELLO[2]

[1]UFMS – Federal University of Mato Grosso do Sul
FACOM – Faculty of Computing
P.O. Box 549, 79070-900 Campo Grande (MS) – Brazil
renato@facom.ufms.br,

[2]USP – University of São Paulo
ICMC – Institute of Mathematics and Computer Sciences
P.O. Box 668, 13560-970 São Carlos (SP) – Brazil
mello@icmc.usp.br

Abstract. The data Grid, a class of Grid Computing, aims at providing services and infrastructure to data-intensive distributed applications which need to access, transfer and modify large data storages. A common issue on Data Grids is the data access optimization, which has been addressed through different approaches such as bio-inspired and replication strategies. However, few of those approaches consider application features to optimize data access operations (read-and-write). Those features define the application behavior, which supports the optimization of operations, consequently, improving the overall system performance. Motivated by the need of efficient data access in large scale distributed environments and by the affordable improvements of application characteristics, this paper proposes a new heuristic to optimize data access operations based on historical behavior of applications. Throughout experiments we concluded that applications are better optimized by anticipating different numbers of future events, which vary over the execution. Then, in order to address such issue, we proposed an adaptive sliding window which automatically and dynamically defines how many future operations must be considered to improve the overall application performance. Simulations were conducted using the OptorSim simulator, which is commonly considered in this research field. Our experimental evaluation confirms that the proposed heuristic reduces application execution times up to 50% when compared to other approaches.

Keywords: data access optimization, grid computing, cluster computing, optimization algorithms, resource allocation, modeling and simulation.

(Received February 23rd, 2011 / Accepted May 2nd, 2011)

1 Introduction

The availability of low-cost microprocessors and the computer network evolution have made feasible the development of distributed systems. In such systems, processes communicate one another in order to perform the same computing task. Besides reducing costs, those systems are scalable and more flexible than real parallel machines [7].

Concepts of distributed systems motivated the development of cluster computing, where resources are usually interconnected in a local area network [26]. These cluster environments have encouraged researches

on process scheduling optimization, data prefetching, distributed file systems, fault tolerance and security. As those systems became more available as well as the Internet has allowed the access of long-distance resources, scientists started interconnecting them to solve more complex problems [33]. That approach started the development of grid computing environments, in which resources are usually heterogeneous, geographically distributed and accessible to several users [33]. The intrinsic features of these platforms have required new researches on job scheduling, data access, fault tolerance and security strategies [33]. Besides the evolution of every topic, the Data Access Problem (DAP) is still a major concern when dealing with grids, mainly due to data location and consistency [8]. Other issues that should be considered are the data migration, replication, distribution and also the evaluation of the data access impact on job scheduling approaches [11].

Besides considering the aforementioned topics, related works (Section 2) present three main drawbacks. The first is that most of the works consider exclusively read-only operations [5]. Such situation tends to restrict the execution of real-world applications. They mainly neglect such subject due to the complexity involved in keeping data consistent. The second drawback is that many works consider static data access optimization approaches, i.e. they do not adapt themselves according to dynamic grid computing features, such as users logging into and out, computers connecting and disconnecting. The last drawback is that the overall system performance depends on data access patterns, what varies according to job system calls. In this way, it is very important to understand, estimate and/or predict the job behavior as a way to optimize read-and-write operations.

The three main drawbacks of related works have motivated an initial investigation [19], in which we evaluated a heuristic to optimize job read-and-write operations on distributed environments, based on applications historical behavior. In that paper, we optimized next file access operations by considering a window of future events (every event corresponds to a file access operation). However, in that study, we considered fixed-length windows and confirmed they result in an unsatisfactory response time to grid environments, mainly due to the high heterogeneity of computational resources (CPUs, hard disks and networks) and the variation of reading and writing operations during the process execution. By considering initial experiments presented in that paper, we observed that the interposition of read-and-write events (i.e., events of different types under the same window length) affects the efficiency of the window. Thus, we were motivated to investigate whether a window of future events under the same type of operation could improve data accesses and, therefore, reduce costs.

Based on such hypothesis, this paper proposes an adaptive sliding window to optimize data access by improving replication, migration and consistency decisions. An adaptive window is defined over the historical behavior of applications, and it represents the number of future events that our heuristic analyzes to optimize decisions. The adaptive window constantly adjusts its length according to the dynamic behavior of processes, i.e., the number of similar operations (readings or writings) under execution. As this approach considers historical job behavior, applications need to be executed at least once.

The specific contributions of this paper are: 1) the formalization of the Data Access Problem (DAP); 2) an analytical optimization model to address DAP, aiming at minimizing the overall application execution times; 3) proposal of an adaptive sliding window approach to define how many historical events are considered in the optimization process. 4) simulations to evaluate the efficiency of the adaptive sliding window under a wide range of environments and system configurations (how read-and-write operations are distributed over time).

Besides all the listed contributions, experimental results confirm that this new adaptive heuristic outperforms other commonly considered ones (e.g., LRU, LFU and Economic Model, presented in Section 5) in approximately 50% when dealing with grid environments.

This paper is organized as follows: Section 2 reviews related work; Section 3 models the Data Access Problem (DAP); the proposed adaptive heuristic is presented in Section 4; Section 5 presents simulation results and, finally, concluding remarks.

2 Related Work

Several studies have been conducted to improve data access on grid environments. Such works are mainly focused on data replication, distribution and consistency.

Oliker et al. [27] propose a static data allocation approach and three data-oriented job scheduling algorithms (SI, RI, SYI). The approach attempts to optimize the system overall performance by allocating jobs where data is available. Among the evaluated scheduling algorithms, SI reduces the average execution time by 60% when compared to local approaches, and can execute 40% more jobs.

Rahman et al. [28] present a model which uses a simple data-mining approach (K-Nearest Neighbor,

KNN) to select the best replicas from grid sites. To select the best replica, the authors design an optimization technique that considers the network latency and the disk state. Different file access patterns are investigated and compared to the KNN algorithm. KNN shows a performance improvement for sequential and unitary random file access patterns.

Sun & Xu [34] propose two consistency algorithms: Lazy-Copy (LC) and Aggressive-Copy (AC). LC updates replicas only when needed, i.e., when a user requires them. This may reduce the bandwidth consumption, avoid unnecessary transfers when data are modified but not required. AC updates replicas whenever a change occurs in the original file. Therefore, AC fully guarantees the consistency, while LC partially guarantees it. In the comparison between the two algorithms, the Aggressive-Copy reduces the access latency, whilst Lazy-Copy reduces the bandwidth consumption.

Wang et al. [36] consider the parallel access of data replicas. The access time is minimized by overlapping requests, what tends to increase the throughput. The proposed solution (called MSDT) carries on replicating data in idle intervals as a way to improve system performance. That work does not consider data consistency. Results present a speedup factor in the range of $2.72 \sim 3.06$ when comparing MSDT to another technique (called NoObserve) [15].

Oldfield & Kotz [26] propose the Armada framework which launches applications and defines how files are distributed. It also provides access control and data access mechanisms. It builds graph structures to represent the processing and data flow. Experiments compare restructured applications to original ones. Armada improves throughput of wide-area networks in 40% when compared to original applications.

Dang & Li [11] propose a tree-type structure to correlate data in grid regions aiming at reducing file transfers. When a job needs a file, the approach looks for high correlation data before asking for transfers. This reduces network costs and improves the overall application performance.

Elghirani et al. [14] define a data management service to replicate files on sites as well as a Taboo Search approach to schedule jobs aiming at optimizing runtime and system utilization. The Taboo Search attempts to find good data replication solutions, considering job data accesses and processing time. Results present performance improvements from 8% to 35%, depending on the replication and job scheduling approaches.

Kim et al. [22] propose a technique to improve data access, matching nodes to the best remote data sources. The authors consider a trace-based synthetic scenario on PlanetLab to evaluate their heuristic. Results show that the resource selection outperforms conventional techniques such as latency-based or random allocations.

Chervenak et al. [10] propose a framework called Replica Location Service (RLS) which maintains and provides information on physical locations of replicas. RLS is used in a variety of production environments such as the Laser Interferometer Gravitational Wave Observatory (LIGO) [3], Earth System Grid (ESG) [1] and Pegasus [12]. Authors presented a performance study demonstrating that the individual RLS servers have performed well and scale up to millions of entries compared to the native MySQL using ODBC clients.

AL-Mistarihi & Yong [4] propose an approach to address the replica selection problem. The authors consider the Analytical Hierarchy Process (AHP) to solve that problem, and they evaluate this approach in an extension of the OptorSim simulator. AHP was employed to solve this optimization problem using a simplification of multiple objectives into a single one. However, the authors evaluated AHP comparing it only against a random approach. They could and should at least compare the performance of AHP against strategies included in the OptorSim simulator, such as LRU, LFU and the Economic Model.

It is important to observe that all previous presented studies do not consider the dynamic behavior of applications when taking decisions, likewise, they do not take advantage of future read-and-write operations to optimize data accesses.

The dynamic behavior of applications motivated Ishii & Mello [19] to propose a heuristic that adopts a fixed length window of future events which aims at anticipating reading and writing operations. Using future events, the heuristic optimizes decisions on replicating, migrating and keeping consistency. This study confirmed that windows of future events can indeed improve application performance in some scenarios. For example, when considering environments with read-only operations, the heuristic improved as much as 100%, however under a low frequency of writing operations (5% of writing and 95% of reading operations) and low frequency of reading operations (95% of writing and 5% of reading operations), the heuristic could not reduce application execution times.

In this previous study [19], the window length is fixed and defined by the system administrator. Based on that we attempted different window lengths and analyzed experimental results, such additional work motivated us to study whether a window of future events, under the same type of operation, could improve data

accesses and, therefore, reduce application execution times. Thus, in this paper, we propose an adaptive sliding window to optimize data accesses by improving replication, migration and consistency decisions.

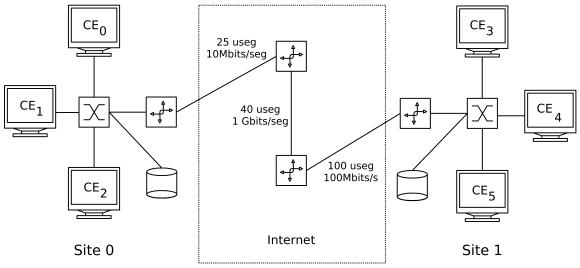

Figure 1: Example of network interconnection

3 The Data Access Problem

This section aims at defining the Data Access Problem (DAP). We start with an empirical case study and then we proceed with the formal definition of DAP.

3.1 An Empirical Case Study

In order to empirically state the DAP, we propose the following hypothetical case study: let two parallel applications be composed of the processes presented in Table 1, where MI represents the million of instructions executed by processes (during their lifecycles); MR and MW are, respectively, the number of KBytes/sec read and write from/into the main memory; HDR and HDW are, respectively, the number of KBytes/sec read and write from/into the hard disk (or secondary memory); NETR and NETS are, respectively, the number of KBytes/sec received and sent through the computer network – in this case, the sender (for NETR) and the receiver (for NETS) processes are presented. For example, Table 1 row 1 shows that process p_0 consumes $1,234$ MI, reads 123 KBytes/sec (and does not write data into memory, i.e. MW$=0$), it still reads 78 KBytes/sec from the hard disk (and it does not write data into the hard disk, i.e. HDW=0), receives 12 KBytes/sec from process p_1 and sends 532 KBytes/sec to process p_1 .

Now consider that these processes are allocated on a set C of computers which is described in Table 2, where CE is a computing element; MIPS represents the processing capacity in million of instructions per second; TMR and TMW are, respectively, the read-and-write throughput of the main memory (in KBytes/sec); THDR and THDW are, respectively, the read-and-write throughput of the hard disk (also in KBytes/sec). Let

Table 1: Processes behavior

App 0

	MI	MR	MW	HDR	HDW	NETR	NETS
p_0	1,234	123	0	78	0	$12 - p_1$	$532 - p_1$
p_1	1,537	23	89	0	12	$532 - p_0$	$12 - p_0$

App 1

	MI	MR	MW	HDR	HDW	NETR	NETS
p_2	1,221	823	70	78	543	$10 - p_3$	$321 - p_4$
p_3	1,137	223	179	324	212	$423 - p_4$	$10 - p_2$
p_4	2,237	23	17	12	0	$321 - p_2$	$423 - p_3$

the allocation operator be defined by $\propto$, where an example of allocation, assuming an application composed of 5 processes, is given by $p_0 \propto \mathsf{CE}_0$, $p_1 \propto \mathsf{CE}_1$, $p_2 \propto \mathsf{CE}_2$, $p_3 \propto \mathsf{CE}_3$ and $p_4 \propto \mathsf{CE}_4$. Computers in C are interconnected according to Figure 1, which also presents an example of network bandwidths and latencies.

Table 2: Grid site characteristics

CE	MIPS	TMR	TMW	THDR	THDW
CE_0	1,200	100,000	40,000	32,000	17,000
CE_1	2,100	120,000	50,000	42,000	19,000
CE_2	1,800	100,000	30,000	22,000	9,000
CE_3	1,700	95,000	20,000	25,000	11,000
CE_4	2,500	110,000	60,000	62,000	30,000
CE_5	2,000	110,000	45,000	40,000	17,000

Also consider that all 5 processes (described in Table 1) of the 2 parallel applications access a set of files $F = \{f_0, f_1, \ldots, f_9\}$. In order to solve the DAP, in an optimal way, we must explore all possible file distributions over the 6 computing elements and evaluate the access cost for every process p_i. In such situation, permutations would be performed to find out all possible solutions. For example, let a solution where the subset of files $\{f_0, \ldots, f_6\}$ is allocated on CE_0 and all others, i.e. $\{f_7, f_8, f_9\}$, are placed on CE_1. The set of all possible solutions, for any instance of the problem, is obtained by computing n^z, where n is the number of computing elements and z is the number of files. Consequently, in the previously mentioned instance, the solution space is equal to $6^{10} = 60,466,176$.

Besides the presented example, it is necessary to study the problem in real-world conditions. For illustration purposes, assume that the target environment contains more than 256 computers. In such situation, the

expected problem solution space would follow the orders of magnitude defined in Table 3, in which we expect to address thousands of computers storing millions of files.

Table 3: Solution space to distribute files over computing elements

CE's	**# files**	**Solutions:** n^z
256	320	$\approx 4 \times 10^{770}$
256	3,200	$\approx 2 \times 10^{7706}$
256	32,768	$\approx 1 \times 10^{78913}$
512	320	$\approx 9 \times 10^{866}$
512	3,200	$\approx 4 \times 10^{8669}$
512	32,768	$\approx 2 \times 10^{88777}$
1,024	320	$\approx 1 \times 10^{963}$
1,024	3,200	$\approx 9 \times 10^{9632}$
1,024	32,768	$\approx 3 \times 10^{98641}$

3.2 Formal Definition

A grid computing environment can be represented as a non-directed graph $G = (S, L)$, where the set of vertexes S represents the grid sites (networks of workstations, parallel machines, clusters, etc.) and the set of edges L is composed of communication links in between grid sites. Also consider a set F containing z files, whose sizes are modeled by function $\theta(.)\colon F \rightarrow \mathbb{Z}^+$. Each grid site $s \in S$ has none, one or more elements with the storage capacity defined by $\Omega\colon S \rightarrow \mathbb{Z}^+$ which is the size in MBytes of the storage element into a grid site s. Function $\alpha_{j,i}\colon F \times S \rightarrow \mathbb{Z}^+$ defines the cost of storing file j on site s_i. In the same way, the function $\delta(.)\colon F \times S \rightarrow \mathbb{Q}^+$ considers the links in L to model the communication cost in between grid sites. Function α is constrained such as $\alpha_{j,i} \leq \Omega_i \ \ \forall j, i$.

Consider the set A which contains k parallel applications executing on a large scale distributed environment. Let the function $\phi(.)\colon A \rightarrow \mathbb{Z}^+$ represent the number of processes of each parallel application. Consider set P which contains all processes of all k parallel applications, i.e. the number of terms in P is equal to $h = \sum_{a \in A} \phi(a)$. All processes are previously allocated on the distributed system according to any scheduling criterion. Every element in P has a set R associated, which contains m read-or-write file requests. A process contains particular features, here defined as behavior, such as processing, memory and input-and-output utilization. Every process, consequently, requires different amounts of resources provided by set S of grid sites as well as the set of communication links L. A process $p \in P$ may access none, one or many files in F.

The cost to transfer file f in between two grid sites s and s' is modeled in Equation 1. It depends on file size $\theta(f)$ and distance $d(s, s')$ in between both sites, which is measured in terms of the network latency and bandwidth. Cost ψ is assumed when a constraint forbids the replication of file f to grid site s'.

$$\delta(s, s') = \begin{cases} \theta(f_j) d(s, s') & \text{where } s \in S,\ j = 1, \ldots, z \\ \psi = \infty & \text{otherwise} \end{cases} \tag{1}$$

A file f must be transferred from the shortest path site $s \in S$ whose cost is minimum and defined by function $d(s, s')$. The access cost for reading a file f, stored in a site s, is, therefore, given by Equation 2. Write operations induct replica updates for every file $f \in F$, which consumes resources as described in Equation 3.

$$r_{\text{cost}}(f) = \delta(s', f) = \arg\min(\theta(f) d(s, s')) \ \forall\ s \in S \tag{2}$$

$$w_{\text{cost}}(f) = \sum_j \delta(s', f_j) \ \forall j \text{ local and remote replicas} \tag{3}$$

By unifying Equations 2 and 3, the total cost to access a file f is determined in Equation 4, considering all processes in P. Table 4 describes each model parameter, afterwards the data access problem is formalized.

$$\Lambda(f) = \sum_{p \in P} \sum_j r_{\text{cost}}(f_j) + w_{\text{cost}}(f_j) \tag{4}$$

Consider a set P of processes which were previously scheduled on grid sites in S with storage capacities defined by α and transfer costs by $\delta(s, s')$. Let a set of files F with a given initial file distribution x_{ij} and size $\theta(f)$. Assume a set of quintuples TR which describes read-or-write operations on files in F. The optimization problem, DAP, consists in determining a new file distribution y_{ij} according to the energy function defined in Equation 5, which is constrained according to Equations 6 and 7 and follows the domains: $x_{ij} \in \{0, 1\},\ \ \forall\ i, j$ and $y_{ij} \in \{0, 1\}\ \ \forall\ i, j$.

$$\Gamma(\textbf{DAP}) = \min \sum_i^n \sum_j^z (x_{ij} - y_{ij}) \Lambda(f_j) \tag{5}$$

Table 4: Model parameters

Parameter	Description
G	The graph of environment $G(S, L)$
S	Set of grid sites
s	Grid site which contains computers, workstations, nodes or similar
n	Number of grid sites
L	Set of communication links
l	Data link in between sites, e.g. $\{s, s'\}$
$\alpha(s)$	Cost function to store data on site s
$\delta(s, s')$	Communication capacity in between s and s'
$\theta(f)$	Size of file f
Ω_i	Total storage capacity of site s_i
A	Set of parallel applications
a	An application
k	Number of applications
$\phi(a)$	Function which determines the number of processes of application a
P	Set of processes
h	Total number of processes
R	Set of read-and-write requests
r	Read-or-write request to a file f
m	Total number of requests
F	Set of files
f	A file
z	Total number of files
i	Grid site index in S ($s_i \in S$)
j	File index in F ($f_j \in F$)
x_{ij}	Equals to 1 if f_j is stored on site at index i. 0, otherwise
y_{ij}	Equals to 1 if the new solution allocates f_j on the site at index i. 0, otherwise
TR	Set of quintuples to describe read-and-write operations to files in F
tr	A quintuple in TR
u	Total number of quintuples

$$\sum_{i}^{n} x_i f_j \leq n \tag{6}$$

$$\sum_{j}^{z} y_{ij} = 1, \;\; i = 1, \ldots, n \tag{7}$$

The energy function (Equation 5) attempts to reduce the cost to request files, considering the distance in between grid sites, amount of data and storage capacity. The constraint presented in Equation 6 limits the number of file replicas, which can not surpass n sites, i.e. if $j = n$ all storage elements in the grid have one replica of f each. The constraint given by Equation 7 defines that every replica must be allocated on one site only. This constraint implies that storage elements do not have more than one replica of file f. All computing elements connected to a grid site access the replica on the storage element of that grid site.

In order to prove that the data access problem is NP-complete, we demonstrate that it is contained in the NP set and it is NP-hard. To demonstrate that the DAP is in NP, a reduction is conducted from the allocation of multiple copies of the same file on a distributed environment (here called Multiple File Allocation (MFA)) which is proven to be NP-complete according to [16]. The MFA problem is defined as:

Instance: *Graph $G(V, E)$, for each $v \in V$ a usage $u(v) \in \mathbb{Z}^+$ and a storage cost $s(v) \in \mathbb{Z}^+$, and a positive integer K.*

Question: *Is there a subset $V' \subseteq V$ such that, if for each $v \in V$ we let $d(v)$ denote the number of edges in the shortest path in G from v to a member of V', we have:*

$$\sum_{v \in V'} s(v) + \sum_{v \in V'} d(v) \times u(v) \leq K? \tag{8}$$

Theorem 1. *DAP is NP-complete.*

Garey & Johnson [16] present three different approaches to prove NP-complete problems: restriction, local replacement and component design. Consider a problem $\Pi \in NP$, the proof using the restriction approach consists in showing that Π contains a known NP-complete problem Π' as a special case. The main idea lies in the specification of additional restrictions on instances of Π, so that the resulting problem will be identical to Π'. The problem Π does not need to be exactly the same as Π', but it must preserve yes-and-no correspondence in their outputs. In local replacement, we must identify the components, or building blocks, which integrate the instance of a known NP-complete problem Π'. In order to prove that a problem Π is

NP-complete, we look for similarities of the Π' basic components and relate them to the Π problem. The last type of proof is the component design, which considers the constituents of the target problem instance to design components to be combined and represent the instance of the already known NP-complete problem.

Proof. In this paper, we prove that DAP is NP-hard by using the restriction-proof approach. DAP is contained in NP due to it is possible to build a non-deterministic machine to verify the graph $G(S, L)$ in polynomial time. When building the machine, the following is non-deterministically defined: for each $s \in S$ an allocation x_{ij} in which the file f_j is mapped, and, for each $l \in L$ the cost $\alpha(s, s')$. The verification step checks out, for each $f \in F$, whether the grid site mapping is valid, this is, if $\sum_{j}^{n} x_j \leq n$. Finally, the condition $\sum_{j}^{n} y_{ij} = 1, \quad i = 1, \ldots, n$ is evaluated for the new allocation of files in F.

Consequently, an instance of DAP is translated into a MFA one. We initially map every element of the set V in elements in S. Then, we define the number z of files. Those files are initially allocated on sites in S, in a random way. This file allocation respects the constraint that each grid site $s \in S$ has the maximum storage capacity of $s(v)$. Besides that, $d(v)$ is mapped into $d(s, s')$ what represents the cost to transfer file $f \in F$ in between two grid sites s and s', and the function $u(v)$ is mapped into $\theta(f)$, which models the file size. For every element $e \in E$, there is a correspondent $l \in L$ with network latency and bandwidth associated.

In the MFA problem, Equation 8 formalizes the objective function which aims at minimizing costs related to the allocation of multiple copies of the same file. This equation is bounded by K. Similarly, in the DAP problem, the objective function is defined by Equation 5 which also attempts to minimize access costs.

Finally, consider an instance for the DAP where: given a request $r \in R$ which is launched by a process $p \in P$ when reading or writing on a file $f \in F$. For any instance of DAP where $|S| \geq 2$ and $|L| \geq 2$, the problem presents exponential characteristics and it is considered NP-complete.

□

4 Data Access Approach

After stating the DAP, we may address it by using exact or approximation approaches. Exact approaches guarantee optimal solutions for the problem. However, they may be very time consuming, depending on the problem and the instance. For small instances, they might offer acceptable run-time, but, for large instances, they are prohibitive [35]. This fact motivated the development of approximation strategies considering heuristics and metaheuristics. A heuristic is an algorithm that, based on the problem knowledge or experience, leads to appropriate solutions, but there is no guarantee to obtain optimal solutions [16].

Heuristics are commonly considered due to their trade off in between the solution quality and the time complexity. Metaheuristic is a type of heuristic to solve a class of problems and not only a specific one. Examples of metaheuristics are: Genetic Algorithms (GA) [18], Ant Colony Optimization (ACO) [13] and Simulated Annealing (SA) [23].

Genetic algorithms have been used as search and optimize techniques in several domains [30]. They are based on the natural selection theory, which guarantees the survival of the most adequate individuals, i.e. the ones that represent good solutions. This approach does not ensure optimal solutions for all problem instances, but provides appropriate solutions for a reasonable number of NP-Complete problems [30].

ACO-based algorithms support the search for paths in between a given source and a destination. This bio-inspired approach is based on the stigmergy strategy and the pheromone concentration. Stigmergy is a communication mechanism used by ants to coordinate global functions. As ants randomly walk looking for food, they lay down pheromone (chemical component released by ants) on trails. When another ant is looking for the food path, it has a certain probability to follow the previous crossed path (according to the pheromone concentration). This approach iteratively reinforces good paths, supporting the search for shortest-path solutions on graphs.

Simulated Annealing aims at finding a global minimum for a given energy function [23]. This nomenclature comes from an analogy to the metallurgic process of annealing, which consists of the controlled heating and cooling of materials as a way of finding stable energy states. Those states, for instance, help metallurgic processes to reduce physical defects on different materials. This technique introduces the system temperature concept, which defines the annealing scheduling (heating and cooling operations). SA supports the search of global minimal of energy functions.

After stating the DAP, we may address it by using metaheuristic, which is a type of heuristic to solve a class of problems and not only a specific one. Examples of metaheuristics are: Genetic Algorithms (GA)

[18], Ant Colony Optimization (ACO) [13] and Simulated Annealing (SA) [23]. However, one issue prevents the adoption of such techniques, which is the dynamic behavior of data accesses. Those metaheuristics would model the DAP by evaluating the current accesses as well as future ones. However, their objective functions would compose such information in the current moment, in a way that they tend to optimize the average behavior. Therefore, such approaches may privilege remote accesses while they could improve the system overall performance by replicating data and supporting local operations. Better solutions could be obtained by assessing behavior changes and using them to anticipate data operations. In order to develop such method, we must identify the dynamics involved in the process behavior.

Given the disadvantages presented by the metaheuristics, we propose a novel heuristic to approach the data access optimization using historical application behavior and the adaptive sliding window length. As the heuristic considers the historical process information, we can anticipate process read-and-write operations and, therefore, replicate data locally before needed, what tends to reduce access costs. When data is locally available, read-and-write operations execute faster, what avoids access delays. The anticipation of process events is used to take decisions on data replication, migration and consistency.

The proposed approach follows the workflow defined in Figure 2, which is composed of the modules: 1) application knowledge acquisition; 2) adaptive sliding window length; and 3) the Heuristic. The following sections describe the features of each module, including their integration.

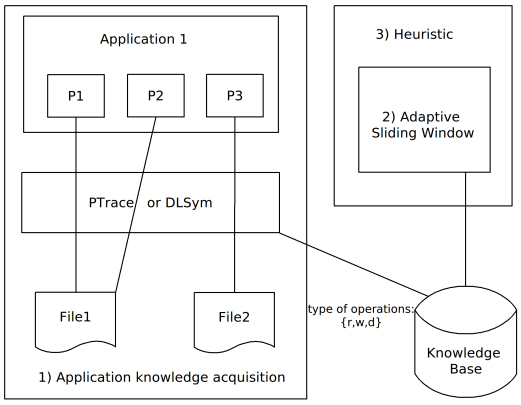

Figure 2: The proposed approach

4.1 Application Knowledge Acquisition

This paper considers applications behavior to guide a novel heuristic to improve decisions on replication, migration and consistency of files. The behavior is composed of read-and-write operations issued by application processes.

There are two approaches to extract process behavior: monitoring and event interception. Monitoring periodically requests information of a given system. Examples of monitoring tools are SMART (Self-Monitoring Analysis and Reporting Technology) [31], Linux Vmstat [37] and Tcpdump [20]. System monitoring is usually based on counters. Counters are variables which account the system utilization within specific time intervals. For example, consider SMART. Let it request the number of hard disk reads and obtain 0 at a given instant. After 10 seconds, it asks for the information again and the system returns 5, what means that five read operations were executed in between the first and the second monitoring points. The monitoring approach grabs the amount of data and what kind of event has happened in a certain time interval, but it does not inform when exactly an event happened and its detailed information. The second approach intercepts process events and calls a procedure to inform about them (this approach obtains the exact event moment as well as its detailed information). However, depending on the event complexity (such as process system calls) and cost, the interception may become prohibitive. Examples of the intercepting tools are the Unix DLSym [21] and Ptrace [32].

DLSym allows the interception of dynamic procedure calls (from dynamic libraries). This is, when the program calls a function, instead of having the code internally, it loads a shared library and runs the procedure. This approach avoids procedure rewriting and also allows the interception of calls. Consequently, any procedure in a dynamic shared library can be intercepted, what helps to build monitoring tools. On the other hand, Ptrace transparently intercepts process signals and system calls. It is usually employed to build diagnosis and debug tools. Ptrace does not use dynamic libraries and can intercept any Unix application.

Researchers must evaluate the options in between monitoring and intercepting and choose the best approach to address the system under study. Some systems can only be monitored, while others, only intercepted. Given the physical characteristics, hardware are usually monitored, while software can be monitored or intercepted. In this paper, the interception approach was chosen due to it allows the continuous extraction of application behavior over time. Furthermore, the monitor-

ing approach may hide behavior in between sampling intervals.

After extracting the application behavior, we transform the sequence of events in series of numerical values which represent time instants. Every event is described by the quintuple $\text{tr} = \{\texttt{pid}, \texttt{inode}, \texttt{amt}, \texttt{time}, \texttt{op}\}$ where `pid` is the identifier of the process that performs the operation, `inode` is the file identifier, `amt` is the amount of data read or written, `time` is the time interval in between operations and `op` is the operation type (whether reading or writing). A series for u sampling intervals is, therefore, defined as $\text{TR} = \{\text{tr}_0, \text{tr}_1, \ldots, \text{tr}_{u-1}\}$ which is used to analyze relations among events. Those events are included in a trace file for future usage. Table 5 presents an example of a trace file.

Table 5: Example of trace file

index	pid	inode	amt	time	op
1	p_{41}	f_0	313	24	**r**
2	p_{89}	f_3	94	171	**w**
3	p_{10}	f_9	92	80	**w**
4	p_{32}	f_0	826	132	**w**
5	p_{69}	–	–	76	**d**
6	p_1	f_5	292	70	**w**

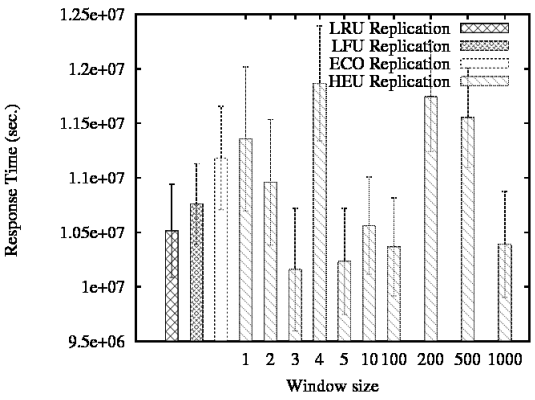

Figure 3: The impacts of sliding window variation

4.2 Adaptive Sliding Window Length

The sliding window length defines the number of future events that the heuristic analyzes to optimize decisions, which impact on the system performance. An example of the impact the sliding window has on applications execution is shown in Figure 3, that considers a sliding window approach with fixed length. Four techniques were evaluated: LRU, LFU, Economic model (see Section 5), and the heuristic with fixed window length (label HEU in Figure 3) [19].

We observe that, for the sliding window length 4, the application execution time is higher. We observed this fact occurs due to the heterogeneity and the interposition of writing and reading operations. This also happens for sliding windows 200 and 500, where there is also high heterogeneity in terms of operations. We experimentally confirmed that the more homogeneous is the execution order of reading and writing operations (i.e. similar operations are arranged consecutively), the longer can be the prediction horizon, i.e. more future events can be considered and, therefore, we can set larger window lengths.

This fact motivated this work that proposes of an adaptive sliding window approach which considers the type and number of operations. Thus, we proposed parameter β to adapt the window length according to consecutive homogeneous operations (i.e. reading or writing). Equation 9 defines the Adaptive Sliding Window (ASW), where W_{t+1} is the next window length, W_t is the current window length, Op is the number of homogeneous operations and β is the factor which determines modifications in the window length.

$$W_{t+1} = W_t \times (1 - \beta) + \text{Op}^{2 \times \frac{\text{Op}}{W_t}} \times \beta \qquad (9)$$

For example, given $W_1 = 10$, $\text{Op} = 4$ and $\beta = 0.10$, we would obtain a next window length equals to $W_2 = 9$. Note that the window adapts according to the number of operations under the same type, see Figure 4. On the other hand, when Op is equal to W_t (see W_4 in Figure 4) the next window length increases considerably, $W_5 = 11$. Parameter β is experimentally defined (as presented in Section 5.4).

Figure 4 presents a sample trace file with process behavior information (in this example we focused only in reading (r) and writing (w) operations) obtained through the interception approach (Section 4.1). The first part of the Figure 4 shows the initial window length ($W_1 = 10$) and all consecutive operations. The second part shows how to compute of the next sliding window length based on Equation 9. Finally, the last part shows the subsequent windows $W_2 = 9, W_3 = 8, W_4 = 7$ and $W_5 = 11$, which clearly demonstrates the adaptation of window length according to the behavior of reading and writing operations.

Initializing the Adaptive Sliding Window

$$\overbrace{rrrrwwrwwr}^{W_1=10}rwrrwwrwwwwwwwrrwrrrrrrr$$

Calculating window length

$$\underbrace{rrrr}_{\text{Op=4}}wwrwwr \rightarrow W_2 = 10 * (1-\beta) + 4^{2*\frac{4}{10}} * \beta = 9$$

$$\underbrace{r}_{\text{Op=1}}wrrwwrww \rightarrow W_3 = 9 * (1-\beta) + 1^{2*\frac{1}{9}} * \beta = 8$$

$$\underbrace{wwwww}_{\text{Op=5}}rrw \rightarrow W_4 = 8 * (1-\beta) + 5^{2*\frac{5}{8}} * \beta = 7$$

$$\underbrace{rrrrrrr}_{\text{Op=7}} \rightarrow W_5 = 7 * (1-\beta) + 7^{2*\frac{7}{7}} * \beta = 11$$

Window length after executing the ASW approach

$$\overbrace{rwrrwwrww}^{W_2=9}\overbrace{wwwwwrrw}^{W_3=8}\overbrace{rrrrrrr}^{W_4=7}\overbrace{rwrrrwwrrwr}^{W_5=11}$$

Figure 4: Example of the Adaptive Sliding Window with $\beta = 0.10$.

By using this mechanism, we compute the number of similar operations and define the length of the next window of future events. Thus, it limits how many future events will be analyzed. Then, we consider the window to anticipate data transfers, bringing information locally and, therefore, reducing future access costs.

Without the adaptive mechanism, we should set a fixed length for the sliding window, which is only possible experimentally (requiring the evaluation of a range of values for the length). Besides that, a fixed length might be good for part of the application execution, while it would reduce the performance in other situations (time instants). Therefore, the length needs to be dynamically adapted according to the instantaneous behavior of the process (we consider one specific adaptive window per process). Moreover, using the adaptive mechanism, the window fits the process behavior over time, making it more flexible and efficient.

4.3 The Proposed Heuristic

Let $P = \{p_0, p_1, \ldots, p_{h-1}\}$ be the set of all processes previously scheduled on grid sites, $F = \{f_0, f_1, \ldots, f_{z-1}\}$ be the set of files to be accessed, and $O = \{r, w, d\}$ (where r is read, w is write and d represents idle moments) represent the set of types of operations. Let TR model the series of process behavior, where each element tr $\in$ TR describes an operation op over a file f, executed by process p.

Thus, we anticipate data access operations over TR by using an adaptive sliding window ASW $\rightarrow [t_y, t_x]$ (considering $x > y$). This sliding window models the number of future events (process behavior) considered when optimizing process data-access operations.

The mechanism of anticipation aims at transferring files before they are requested by jobs in execution. This mechanism reduces the access cost due to files are requested beforehand and they are locally replicated. Consequently, all this scenario reduces the total execution time of applications. In Figure 5, a file will be requested at t_1, and it was totally transferred in advance. On the other hand, in Figure 6, the file is requested in t_1, however the transfer was not completed and, in this case, the heuristic still needs to transfer the spare data. In this last scenario, the total cost is $t_2 - t_1$.

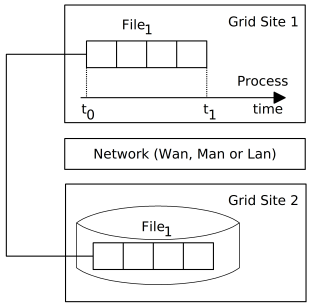

Figure 5: The file is completely transferred

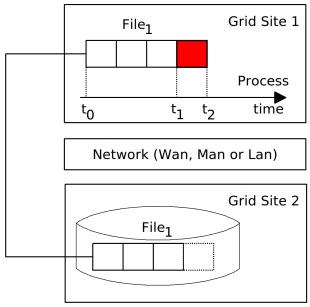

Figure 6: The file is partially transferred

As previously presented in Table 5, every trace line (event) corresponds to a quintuple tr $\in$ TR. Thus, the proposed heuristic (Algorithm 5) considers the historical process behavior, represented by TR, and a given sliding window ASW of future operations to optimize data accesses on distributed environments. The anticipation of future operations supports the minimization of communication/synchronization costs related to file transfers and consistency.

In this way, consider a process p which executes operations over a file f. Now, let p be at the current execution or time instant t_c and there is a window of events up to the time instant t_f, where $t_f > t_c$. The proposed heuristic evaluates if file f can be retrieved (i.e. to find the latest version of a replica f') to the local site where it is needed. If so, transfer costs and access delays are reduced and the process performance is improved. In this circumstance, there are two ways of retrieving f, both imply cost Γ (Equation 5):

1. if $t_c + \Gamma \leq t_f$, then we can completely retrieve file f, which reduces the process execution time, in Γ units, to access it;

2. if $t_c + \Gamma > t_f$, then we can reduce $t_f - t_c$ time units in overall process execution.

The proposed heuristic is presented in Algorithm 5 and in other auxiliary procedures as follows (procedures listed in Algorithms 1, 2, 3 and 4).

The method `Retrieve`(f) (Algorithm 1) establishes a new local copy (replication) of the remote file f and assesses the copy removal option (for data migration purposes). The method `Read`(tr) (Algorithm 2) receives a quintuple tr and, if file f is *invalid* then it applies the method `Retrieve`(f) and finally reads it. The method `Write`(tr) works similarly (Algorithm 3). The method `Invalidate`(f) (Algorithm 4) receives a file and updates all replicas as *invalid*.

Algorithm 1 `Retrieve`(f)

Require:
The f file.
Ensure:
f' replica file retrieved from nearest site.

1: $S \leftarrow$ sites subset where there are f copies
2: **for** `each` $s \in S$ **do**
3: $\Lambda(f) = \sum_{p \in P} \sum_j r_{\text{cost}}(f_j) + w_{\text{cost}}(f_j)$
4: **end for**
5: return f'

Every grid site $s \in S$ (where the environment is represented by the graph $G(S, L)$) has an associated set of processes which were previously scheduled according to any policy. The method `getNextProcess`(s) (Algorithm 5, line 3) returns a process p scheduled on site s. Every grid site s contains the historical traces of processes (TR, line 4). The pseudocode in between lines 5 and 13 performs operations tr $\in$ TR, considering the possible types: read, write or idle. From line 14 to 44, the heuristic assesses the feasibility of replication,

Algorithm 2 `Read`(tr)

Require:
Quintuple {`pid`, `inode`, `amt`, `time`, `op`}.
Ensure:
Reading `amt` from file referenced by `inode`.

1: **if** f is "*invalid*" **then**
2: `Retrieve`(f)
3: **end if**
4: reads

Algorithm 3 `Write` (tr)

Require:
Quintuple {`pid`, `inode`, `amt`, `time`, `op`}.
Ensure:
Writing `amt` to the file referenced by `inode`.

1: **if** f is "*invalid*" **then**
2: `Retrieve`(f)
3: **end if**
4: writes

Algorithm 4 `Invalidate`(f)

Require:
The f file.
Ensure:
The subset S of outdated f replicas.

1: $S \leftarrow$ site subset where there are f copies.
2: **for** `each` $s \in S$ **do**
3: update copy c_s of f as "*invalid*" in site s.
4: **end for**
5: return S.

Algorithm 5 DAP Heuristic

Require:
Set of process $P = \{p_0, p_1, \ldots, p_{h-1}\}$;
Set of files $F = \{f_0, f_1, \ldots, f_{z-1}\}$;
Set of quintuples TR $= \{\text{tr}_0, \text{tr}_1, \ldots, \text{tr}_{u-1}\}$;
initialASW $= 100$; ASW > 1; $\beta = 0.10$.
Ensure:
Set of file replicas F'.

```
 1: for each p ∈ P do
 2:   p ← getNextProcess()
 3:   for each tr ∈ TR do
 4:     if p = getProcess(tr) then
 5:       if getOperation(tr) = "Read" then
 6:         Read(tr)
 7:       else if getOperation(tr) = "Write" then
 8:         Invalidate(f)
 9:         Write(tr)
10:       else
11:         Sleep(tr.time)
12:       end if
13:     else if tr.index ≤ u then
14:       j ← tr.index
15:       ASW ← initialASW
16:       k ← 1; Op ← 0; diff ← true
17:       prevOp ← getOperation(tr)
18:       while k < ASW AND j < |TR| do
19:         if p = getProcess(tr_j) then
20:           if getOperation(tr_j) = "Read" then
21:             if f is invalidate OR f is not local AND
                none bringing the data then
22:               Retrieve(f)
23:             end if
24:           else
25:             if f is invalidate OR f is not local then
26:               Retrieve(f)
27:             end if
28:           end if
29:           k ← k + 1
30:           currentOp ← getOperation(tr)
31:           if prevOp = currentOP AND diff then
32:             Op ← Op + 1
33:             prevOp ← currentOp
34:           else
35:             diff ← false
36:           end if
37:         end if
38:         j ← j + 1
39:       end while
40:       initialASW ← ASW × (1 − β) + Op^(2 × Op/ASW) × β
41:       if initialASW < 2 then
42:         initialASW ← 2
43:       end if
44:     else
45:       Sleep(tr.time)
46:     end if
47:   end for
48: end for
```

migration, consistency and retrieval of files, according to the historical events in sliding window. Lines 32 to 34 compute the number of similar operations in the current window.

Line 41 computes the next window length and line 43 is a special case when window length is less than 2. This special case is a restriction which denies the absence of further updates in order to avoid shorter-length windows (which could risk the adaptive approach). Line 46 corresponds to the situation in which site s is not responsible by process p and there is no event in the sliding window, therefore, s remains idle for a period of `time`.

4.4 Complexity evaluation

In order to understand the complexity of the proposed heuristic, we focused on the dominant computations in the Algorithm 1. The outer loop (line 1) is repeated at most P times, therefore, its time complexity is $O(|P|)$. The loop at line 3 is execute, in worst case, |TR| times. Further, for each iteration of the inner loop (line 18), lines from 19 to 38 have the time complexity of $O(|\text{TR}|)$. We concluded that the total time complexity of the heuristic algorithm is $O(P \times \text{TR}^2)$. Thus, our heuristic algorithm is low-order polynomial and can be run quickly for various grid environments with several sites.

5 Experiments

Grid computing researches are based on two types of experiments: real and simulations [11]. Experiments in real environments are usually more reliable and confirm proposed approaches in practice. However, applications may run for long periods and, furthermore, they need to be correctly and functionally implemented. Most likely failures occur during those executions, which influence the results accuracy. Other workloads, imposed on the environment, may also interfere in experiments. Finally, it is very difficult to have fully dedicated environments to avoid those interferences, mainly when considering the inherently large scale of grid computing scenarios.

Simulations approximate real experiments by modeling (using equations) those environments and their iterations (computers, processes, operations, etc.). The advantages of simulations over real experiments are: no need of building real systems; no limits of experimental scenarios; full control of the environment and reproducibility of experiments.

Due to the advantages of simulation, this paper adopts the OptorSim simulator, which is part of the Eu-

ropean DataGrid EDG Project [5]. This simulator was originally developed to model the dynamic effects of data replication, and it is used to model the LCG (Large Hadron Collide Computing Grid), also considered by other works [22, 4, 10].

OptorSim models grid environments by using sites interconnected through communication links (Figure 7). Every grid site contains at least one computing element (CE, or computer), and one storage element, or both. Every grid site executes a local replica optimizer (RO) to take replication decisions and there is a single global resource broker (RB) to decide on job scheduling. OptorSim models jobs to access a set of files, which may be replicated on grid sites, according to the RO. A replica catalog (RC) maps logical names to physical files and a replica manager (RM) handles replications and also registers them in the catalog. Figure 7 presents all components of the OptorSim simulator.

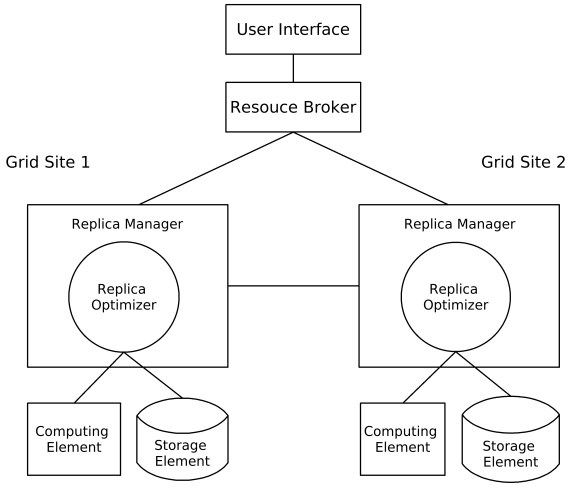

Figure 7: Simulated Data Grid architecture

This simulator allows users to specify the grid topology by providing parameters for CE's and communication links. In this work, we consider a topology generated by BRITE [25], a complex-network topology generator which provides bandwidth and latency information based on well-accepted models. Topology generators are widely used to model large-scale network environments [25], such as the Internet. BRITE was considered due to its flexibility, adaptability and interoperability [25]. The BRITE output file contains communication links with their associated bandwidths and latencies which are used, by OptorSim, to model the grid environment.

OptorSim requires the definition of a job list to be associated to a set of files. Jobs may access a subset of those files, according to an access pattern. The simulator also has four algorithms to take job scheduling decisions: `Random`, `QueueLength`, `AccessCost` and `QueueAccessCost`. `Random` randomly assigns a job to a CE, similarly to Legion scheduling [9]. `QueueLength` assigns jobs to the shortest-queue CE, what may represents the idler resource. `AccessCost` estimates the access time for all files required by a job, then, the job is assigned to the CE with the lowest estimated cost. The last algorithm operates similarly to the latter, but it also considers queue length (hybrid approach).

In addition to the scheduling algorithm, OptorSim provides five access optimization approaches: 1) In `SimpleOptimiser`, no replication is performed, files are remotely accessed; 2) `DeleteOldestFileOptimiser` replicates files when jobs need them, removing the least-recently-used (LRU) replicas; 3) `DeleteLeastAccessedFileOptimiser` replicates files when jobs need them, removing the least-frequently-used (LFU) replicas; 4) `EcoModelOptimiserBinomial` considers an economic model to determine replications. In this approach, file replicas are removed according to a Binomial estimation function; 5) `EcoModelOptimiserZipf` considers an economic model to replicate files. Replicas are removed according to a Zipf estimation function. We employed the approaches LRU, LFU and `EcoModelOptimiserZipf` in the experiments (Section 5.4) in order to compare them with the heuristic proposed in Section 4.3.

OptorSim provides performance metrics for its entities. For grid sites, OptorSim provides: number of remote reads; number of local reads; percentage of time that every CE has been active; number of file transfers that were routed through a site; and the total time (in seconds) consumed to run all jobs submitted to a site. For storage elements, OptorSim provides: capacity and usage, in MB, of the SE. OptorSim also provides the following metrics for computing elements: job execution time, in milliseconds; job execution time added to the queuing time; number of remote file reads of a CE; number of jobs currently executed by a CE; number of local file reads of a CE; percentage of time that a CE was running jobs; list of the files accessed by jobs running on a specific CE; and, finally, the total time (in seconds) to execute all jobs.

5.1 OptorSim Extensions

Some extensions were necessary to adapt OptorSim to meet our needs. Those extensions provide support for:

1) trace files; 2) write operations, and 3) adaptive sliding window of historical operations.

OptorSim provides different file access pattern approaches: 1) `SequentialAccessGenerator`, where files are sequentially accessed; 2) In `RandomAccessGenerator`, files are accessed using a uniform distribution; 3) In `RandomWalkUnitaryAccessGenerator`, files are accessed using a unitary random walk distribution; 4) `RandomWalkGaussianAccessGenerator`, where files are accessed using a Gaussian random walk distribution; and 5) In `RandomZipfAccessGenerator`, files are accessed using a Zipf random distribution. Those access patterns dynamically generate the job behavior, what did not meet our needs to have historical operations. That motivated us to develop a new approach: 6) a trace-based file access approach, where events represent process operations (behavior). Those events provide the following information for every operation: process identification (`pid`), interval in between consecutive access operations (`time`), file identification (`inode`), operation type (`op`) and volume of data (`amt`). Table 5 presents an example of a trace file.

Moreover, the original OptorSim provides only read operations, thus, we also needed to extend it to incorporate write operations. This extension was based on the cost (Equation 3) to write data and propagate it to file replicas, as presented in Algorithms 3 and 4.

Finally, we developed an adaptive sliding window strategy to provide future events (operations) based on historical information, as previously presented in Equation 9. Experiments have confirmed that the adaptive sliding window heavily impacts the heuristic performance (Section 5.4).

5.2 Environment Parametrization

A 128-site grid was considered in experiments. Every grid site was composed of one `SE` where the capacity was modeled by an exponential probability function with average 100GB (parameter Ω described in Section 3). Communication links were modeled by using the BRITE topology generator which considers the Barabasi model for Autonomous System [25]. In addition, BRITE adopts a Heavy-Tail distribution function to define the addition of nodes and bandwidth (the minimum bandwidth was 10 Mbits/s and the maximum $1,024$ Mbits/s). An exponential distribution function was considered to model communication delays (with average 0.5 seconds).

Experiments were configured with 128 jobs (one per grid site). More jobs may be considered, however when a set of jobs runs in a specific grid site, the behavior of each job contributes to the data access behavior issued by that grid site. Then, by having one job per site, we can indeed represent the data access patterns of several individual jobs allocated into the same grid site.

A uniform distribution function (average 10) was used to determine the number of files accessed by every job (i.e. the set of files accessed by a job). Files in the OptorSim have fixed size and can be model by following the chunk representation in the GFS [17]. The behavior of every job is assigned to the simulator by using a configuration file, which may be synthetic or obtained by using an interception approach (Section 4.1 and Table 5). All jobs have the same probability to run in every `CE` of the environment. An exponential distribution function was adopted to model the job inter-arrival time (average $1,500$ ms).

Processing capacities were obtained by using the benchmark SPEC CINT 2000, which is a standard way of measuring CPU performance. It generates a performance measurement based on a reference machine (Sun Ultra 10) [2]. For example, a Pentium IV has a performance of 0.5 CINT 2000. In this work, every `CE` is homogeneous and has a processing power of 1.0k CINT 2000 (or $1,000$ CINT 2000). The CPU homogeneity does not influence in the data access operations which are the main focus of this paper.

We consider the same job scheduling algorithm in every experimental scenario, which is the `QueueAccessCost` (Section 5) and the same trace file to describe the data access pattern of files (Section 5.1).

5.3 Intrusion Analysis when Capturing Information

We evaluated costs involved in acquiring the necessary application knowledge (operations, also called events) to generate trace files and, therefore, assessed the proposed approach. The same cost involved in obtaining such information would be necessary in a real environment. Two acquiring approaches were considered: Ptrace [32] and the DLSym [21] under two benchmarks: `Nbench` [24] and `Bonnie` [6]. `Nbench` probes the CPU capacity in terms of float-point operations as well as the memory subsystem. `Bonnie` performs read-and-write operations on files. Experiments were executed 30 times on a `Intel Core i7 CPU 2.67GHz, 8GB RAM and 250GB HD`. We have evaluated the execution time of such benchmarks with and without acquiring mechanisms. Table 6 presents results which confirm that the intrusion is lower than 12% in the worst case of Ptrace and 1.5% for DLSym. Each value in Table 6 corresponds to the benchmark

execution time and its standard deviation, respectively. This confirms that the second approach, this is DLSym, would better suit on a real environment, that is why we selected it.

Table 6: Information capture approach evaluation

Bench	Time(s)	`Ptrace`	`DLSym`
Nbench	272.81	309.69	276.78
	±5.99	±20.26	±5.46
Bonnie	10.41	11.97	10.47
	±0.03	±0.07	±0.04
Nbench Impact		11.91%	**1.43%**
Bonnie Impact		13.03%	**0.57%**

5.4 Results

This section presents experimental results which are organized according to the previously described environment: 128-site grid. Charts present bars which correspond to the average of 30 executions (this number of executions is based on the Central Limit Theorem [29], which supports the obtainment of a significative statistical measurement).

Experiments are categorized according to the percentage of read-and-write operations. For example, Figure 9 presents an environment with 5% of the writing operations, i.e., the trace file has 5% of events classified as **w** (Table 5).

We evaluated three optimization techniques: LRU, LFU and the Economic Model (ECO). All those techniques are available in the OptorSim simulator. We also compared the results of such techniques against the heuristic proposed is this paper. The DAP heuristic is evaluated under different values of parameter β: $\{0.05, 0.10, 0.15\}$.

In Figure 8, the simulated environment has 100% of reading operations and 10 files are accessed by jobs. The x-axis corresponds to parameter β considered by the heuristic and the y-axis represents the average execution time of jobs (in seconds). As shown in Figure 8, the heuristic was capable of reducing the job execution time in one order of magnitude, when compared to other data replication techniques.

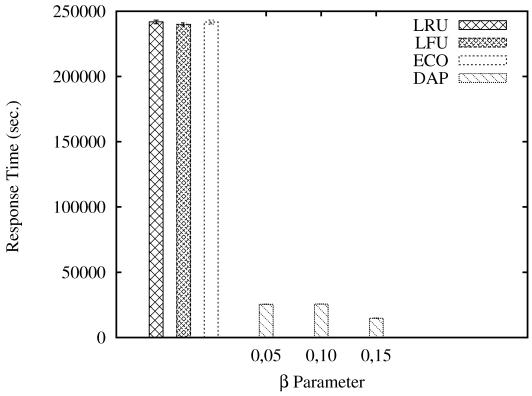

Figure 8: Environment with 100% of reading operations

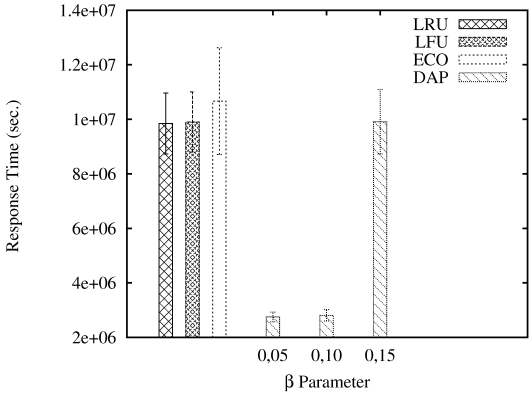

Figure 9: Environment with 5% of writing operations

In Figure 9, the environments deal with 5% of writing and 95% of reading operations and 10 files are accessed by jobs. The heuristic reduced the execution time in one order of magnitude, but, when $\beta = 0.15$, the results are similar to the other techniques.

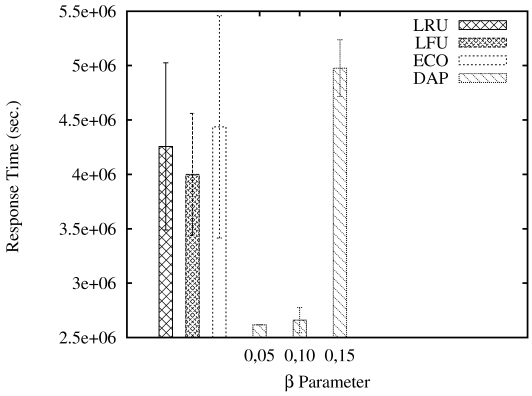

Figure 10: Environment with 95% of writing operations

In Figure 10, the simulated environments present 95% of writing and 5% of reading operations and 10 files are accessed by jobs. The heuristic has reduced job execution times in around 40%. When $\beta = 0.15$, the heuristic results were worse than other replication techniques.

We also compared the results of the adaptive heuristic to the ones obtained in the initial paper [19]. From that, we concluded that this heuristic presents similar execution times to the environments under 100% of reading operations, and this new approach was 10% better for the environment under 5% of writing operations and 15% better for the environment under 95% of writing operations, what confirms the need for an adaptive window.

These results confirm improvements in application performance and suggest the adoption of the adaptive sliding window length. We observed that the adaptation of the sliding window is better when considering $\beta \in [0.05, 0.10]$. In the considered scenario, we experimentally confirmed that $\beta \in [0.05, 0.10]$ gives more relevance to the current window, W_t, according to Equation 9, and when increasing β, more relevance is given to Op. This fact, allow us to conclude that as β gets lower, the adjustment of the window length is improved.

The reduction of access costs was obtained due to the adaptation of the sliding window length (i.e., the window now follows the dynamic behavior of processes), which updates the number of future events considered when taking decisions on data replication, migration and consistency. We also observed that the greater the differences in access patterns (different types of operations – read, write and idle – interposed), the unstabler the application performance is.

6 Conclusions

This paper has presented a history-based data access optimization approach for grid computing environments. Our main objective is to minimize the application execution time by optimizing data accesses and, therefore, improve decisions on replication, migration and consistency. From that, we proposed an adaptive sliding window length which aims at providing the dynamic behavior of application operations to our data access optimization heuristic.

The proposed approach also considers concepts of monitoring and intercepting system calls to capture applications operations and compose processes histories (i.e. the trace files obtained by intercepting calls). We kept such histories due to we believe that it is very important to understand, estimate and/or predict processes behavior as a way to optimize read-and-write operations, which was cleared confirmed. In addition, we defined an analytical optimization model for the Data Access Problem (DAP), which was considered to study approaches for minimizing the overall application execution times.

Simulations were conducted to study the efficiency of our heuristic using the adaptive sliding window length, under a wide range of environments and system configurations (frequency of read-and-write operations). Experimental results confirm that our approach outperforms other commonly considered ones (e.g., LRU, LFU and Economic Model) in approximately 50% when dealing with grid environments. Besides using historical information, the results motivate further work in designing and implementing on-line prediction mechanisms to take autonomic decisions on data replication, migration and consistency.

Acknowledgments

This paper is supported by CNPq-Universal (National Counsel of Technological and Scientific Development), under grant no. 470739/2008-8, CAPES (Coordination of Improvement of Higher Education) and FUNDECT (Foundation to Support the Education, Science and Technology Development of Mato Grosso do Sul) no. 23/200.402/2008 from Brazil. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of CNPq, CAPES or FUNDECT.

References

[1] The Earth System Grid–ESG Project. www.earthsystemgrid.org, 2005.

[2] SPEC's CPU Benchmark. See the definition of CINT2000 at http://www.spec.org/, Mar 2010.

[3] Abramovici, A., Althouse, W. E., Drever, R. W. P., Gürsel, Y., Kawamura, S., Raab, F. J., Shoemaker, D., Sievers, L., Spero, R. E., Thorne, K. S., Vogt, R. E., Weiss, R., Whitcomb, S. E., and Zucker, M. E. LIGO: The Laser Interferometer Gravitational-Wave Observatory. *Science*, 256(5055):325–333, 1992.

[4] AL-Mistarihi, H. H. E. and Yong, C. H. On fairness, optimizing replica selection in data grids. *IEEE Trans. Parallel Distrib. Syst.*, 20(8):1102–1111, 2009.

[5] Bell, W. H., Cameron, D. G., Millar, A. P., Capozza, L., Stockinger, K., and Zini, F. Optorsim: A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[6] Bray, T. Bonnie benchmark. http://www.textuality.com/bonnie/, Jun 2009.

[7] Buyya, R. *High Performance Cluster Computing – Architecture and Systems*, volume 1. Prentice Hall, 1999.

[8] Chang, R.-S. and Chang, J.-S. Adaptable replica consistency service for data grids. In *Proc.* 3^{th} *Int. Conf. on Information Technology: New Generations*, pages 646–651, April 2006.

[9] Chapin, S. J., Katramatos, D., Karpovich, J., and Grimshaw, A. S. The Legion resource management system. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 162–178. Springer Verlag, 1999.

[10] Chervenak, A. L., Schuler, R., Ripeanu, M., Amer, M. A., Bharathi, S., Foster, I., Iamnitchi, A., and Kesselman, C. The globus replica location service: Design and experience. *IEEE Trans. Parallel Distrib. Syst.*, 20(9):1260–1272, 2009.

[11] Dang, N. N. and Lim, S. B. Combination of replication and scheduling in data grids. *International Journal of Computer Science and Network Security*, 7(3):304–308, Mar 2007.

[12] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):25–39, March 2003.

[13] Dorigo, M. and Di Caro, G. The ant colony optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[14] Elghirani, A., Subrata, R., and Zomaya, A. Y. Intelligent scheduling and replication in datagrids: a synergistic approach. In *Proc.* 7^{th} *IEEE Int.Symposium on Cluster Computing and the Grid*, pages 179–182, Washington, DC, USA, 2007.

[15] Feng, J. and Humphrey, M. Eliminating Replica Selection - Using Multiple Replicas to Accelerate Data Transfer on Grids. In *Proc.* 10^{th} *Int. Conf. of Parallel and Distributed Systems*, page 359, Washington, DC, USA, 2004. IEEE Computer Society.

[16] Garey, M. R. and Johnson, D. S. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[17] Ghemawat, S., Gobioff, H., and Leung, S.-T. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[18] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[19] Ishii, R. P. and de Mello, R. F. A history-based heuristic to optimize data access in distributed environments. In *21st IASTED International Conference Parallell and Distributed Computing and Systems (PDCS2009)*, Cambridge, MA, Nov. 2-4 2009.

[20] Jacobson, V., Leres, C., and McCanne, S. TCPDump Man Pages. Available at: Linux Systems, calling the man command for tcpdump, Mar 2010.

[21] Jung, C., Woo, D.-K., Kim, K., and Lim, S.-S. Performance characterization of prelinking and preloadingfor embedded systems. In *Proc.* 7^{th} *ACM & IEEE Int. Conf. on Embedded Software*, pages 213–220, New York, NY, USA, 2007. ACM.

[22] Kim, J., Chandra, A., and Weissman, J. B. Using data accessibility for resource selection in large-scale distributed systems. *IEEE Trans. Parallel Distrib. Syst.*, 20(6):788–801, 2009.

[23] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[24] Mayer, U. F. Linux/unix nbench. http://www.tux.org/˜mayer/linux/bmark.html, Mar 2010.

[25] Medina, A., Lakhina, A., Matta, I., and Byers, J. Brite: an approach to universal topology generation. In *Proc.* 9^{th} *Int. Symposium on*

Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pages 346–353, 2001.

[26] Oldfield, R. and Kotz, D. Improving data access for computational grid applications. *Cluster Computing*, 9(1):79–99, January 2006.

[27] Oliker, L., Biswas, R., Shan, H., and Smith, W. Scheduling in heterogeneous grid environments: The effects of data migration. In *Proc.* 12^{th} *Int. Conf. on Advances in Computing & Communications*, Ahmedabad, INDIA, Jan 2004.

[28] Rahman, R. M., Barker, K., and Alhajj, R. Replica selection in grid environment: a data-mining approach. In *Proc. Symposium on Applied Computing*, pages 695–700, New York, NY, USA, 2005. ACM.

[29] Schefler, W. C., editor. *Statistics: concepts and applications*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1988.

[30] Semenov, M. A. and Terkel, D. A. Analysis of convergence of an evolutionary algorithm with self-adaptation using a stochastic lyapunov function. *Evol. Comput.*, 11(4):363–379, 2003.

[31] SMART. Self-Monitoring, Analysis and Reporting Technology. Available at: http://en.wikipedia.org.wiki/s.m.a.r.t., 2010.

[32] Spillane, R. P., Wright, C. P., Sivathanu, G., and Zadok, E. Rapid file system development using ptrace. In *Proc. Workshop on Experimental Computer Science*, page 22, New York, NY, USA, 2007. ACM.

[33] Stockinger, H. Defining the Grid: a snapshot on the current view. *The Journal of Supercomputing*, 42:3–17, 2007.

[34] Sun, Y. and Xu, Z. Grid replication coherence protocol. In *Proc.* 18^{th} *Int. Symposium on Parallel and Distributed Processing*, pages 232–239, April 2004.

[35] Voß, S. Meta-heuristics: The state of the art. In *ECAI '00: Proceedings of the Workshop on Local Search for Planning and Scheduling-Revised Papers*, pages 1–23, London, UK, 2001. Springer-Verlag.

[36] Wang, C.-M., Hsu, C.-C., Chen, H.-M., and Wu, J.-J. Efficient multi-source data transfer in data grids. In *Proc.* 6^{th} *IEEE Int. Symposium on Cluster Computing and the Grid*, pages 421–424, Washington, DC, USA, 2006. IEEE Computer Society.

[37] Ware, H. and Frederick, F. VMstat Man Pages. Available at: Linux Systems, calling the man command for vmstat, Mar 2010.

INTERPRETOR: A Software Architecture for the Interpretation of Large and Noisy Data Sets

APKAR SALATIAN

School of Information Technology and Communications
American University of Nigeria
Lamido Zubairu Way, Yola By-Pass
PMB 2250, Adamawa State, Nigeria
apkar.salatian@aun.edu.ng

Abstract. In many domains there is a need to interpret the high volumes of noisy data. In this paper we propose and describe a new software architecture called INTERPRETOR for summarising and interpreting voluminous high frequency noisy data. INTERPRETOR consists of 3 modules: Filter which processes noise; Abstraction which abstracts features from the filtered data; and Interpretation which takes the abstractions and provides an interpretation of the data. In this seminal article we also show how INTERPRETOR has successfully been applied to 2 case studies.

Keywords: oftware architecture, filtering, abstraction, interpretation.

(Received February 11st, 2011 / Accepted May 2nd, 2011)

1 Introduction

In many domains there is a need to interpret high frequency noisy data. Interpretation of such data may typically involve pre-processing of the data to remove outliers, inconsistencies or noise. Rather than reasoning quantitatively on a point to point basis which is computationally expensive, this filtered data would be processed to derive abstractions which would be interpreted and the results reported. Such a common approach lends itself to the development of a software architecture.

Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interactions among these components. Such a system may in turn be used as a (composite) element in a larger system design. Indeed, a good software architecture will involve reuse of established engineering knowledge [19].

In this seminal paper we propose and describe the INTERPRETOR software architecture for interpreting and summarising high frequency noisy data sets. INTERPRETOR was inspired by the software architecture of ASSOCIATE [18] for interpreting Intensive Care Unit monitor data and ABSTRACTOR [16] for interpreting building sensor data - both systems have common features which facilitates a generic architecture. INTERPRETOR is based on the pipe and filter software architecture and consists of 3 consecutive processes: Filter which takes the original data and removes outliers, inconsistencies and noise; Abstraction which takes the filtered data and derives abstractions; and Interpretation which takes the abstractions and provides an interpretation and summarisation of the original data.

The structure of this paper is as follows. Section 2 discusses related work. Section 3 describes the INTERPRETOR software architecture to interpret and summarise high frequency noisy data Section 4 describes how the INTERPRETOR software architecture has been applied to 2 case studies. Final conclusions are given in section 5.

2 Related Work

A common architecture used to interpret high frequency data is the *blackboard* as used by ([1],[10], [14]). A blackboard system consists of a set of independent modules, called Knowledge Sources (KSs) that contain the domain-specific knowledge in the system, and a blackboard which is a shared data structure to which all the KSs have access. When a KS is activated it examines the current contents of the blackboard and applies its knowledge either to create a new hypothesis and write it on the blackboard, or to modify an existing one [13]. INTERPRETOR's architecture is similar to that of the blackboard in that individual tasks are performed by separate processes. However since the knowledge of blackboards are distributed, problems would arise with the co-ordination of knowledge sources which have competing obligations.

Another approach for interpreting high frequency and noisy data is the *service-oriented architecture (SOA)* used by ([4], [8], [20]). The SOA consists of components which can handle numerous distributed agents to allow the interpretation of data. Due to the architecture of SOA, extra functionality in the form of security for message passing between agents has had to be incorporated. Since INTERPRETOR is not a distributed system, there is no requirement for these extra services.

Another approach is a *multi-layered architecture* ([7], [21]). A multi-layered system is organized hierarchically where each layer provides service to the layer above it and serves as a client to the layer below it. The authors of [7] proposed a five layered generic and scalable architecture which uses components, middleware and agent technologies. Though a demonstrator was developed as a proof that the proposed conceptual software architecture is feasible in practice, there was no actual data or results to support their design. In contrast our proposed software architecture has been proven to work effectively.

Another approach is to have a *multi-agent architecture* ([2], [3], [12]). An agent is an autonomous computational process that inhabits an Agent Platform. An Agent platform provides the physical infrastructure in which agents are deployed and consists of the machines, operating systems, agent management components, the agents themselves and any additional support software. Agents typically offer one or more computational services that can be published as a service description. Agents typically communicate with each other to fulfill a task. Again, since INTERPRETOR is not a distributed system, there is no requirement for extra services which a multi-agent system would require.

A *non-hierarchical architecture* is SIMON [5]. SIMON (Signal Interpretation and MONitoring) is a system for monitoring neonates in the ICU. SIMON consists of a number of modules implemented as independent UNIX processes, communicating with each other through an inter-process communication (IPC) message protocol. The problem with such an architecture is the scheduling of the various modules. Care must be taken that the shared memory does not get corrupted by simultaneous writes. Another problem of SIMON is that it is solely an event driven architecture - it functions with discrete and infrequently determined input. INTERPRETOR deals with continuous and discrete data and does not have scheduling issues.

Another non-hierarchical approach is to use sequential processes. VIE-NET [11] a monitoring and therapy planning system for the artificial ventilation of newborn infants and resembles a em pipe and filter architecture where by a component reads streams of data on its inputs and produces streams of data on its outputs for another component to process. VIE-NET is made up of four sequential modules: *Data Selection* which filters out context-relevant data for further analysis; *Data Validation* which arrives at reliable measurements by detecting faulty data; *Data Abstraction* which transforms quantitative data of the observable system into qualitative values; and *Data Interpretation and Therapy Recommendation* which performs patient state assessments and associated therapy advise. INTERPRETOR strongly resembles VIE-NET except that INTERPRETOR does not perform data validation (though it could be performed as part of the Interpretation module) nor does it generate therapy recommendations because it is a generic architecture

3 Software Architecture

Figure 1 shows the Context Diagram of the INTERPRETOR system. The INTERPRETOR system takes high frequency noisy data and other relevant data to assist in interpretation from various input sources and presents to various output sources an interpretation of the original data.

Figure 2 shows the data flow in the INTERPRETOR system of Figure 1. Data is initially filtered to get rid of noise; rather than reasoning on a point to point basis, the resulting data stream is then converted by a second process into abstractions - this is a form of data compression. These abstractions are interpreted by a third process to provide an assessment of the original data.

We, therefore, derive the overall software architecture of the INTERPRETOR System in form of a Structure Chart as shown in Figure 3.

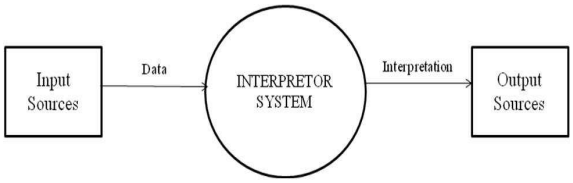

Figure 1: Context Diagram of the INTERPRETOR System

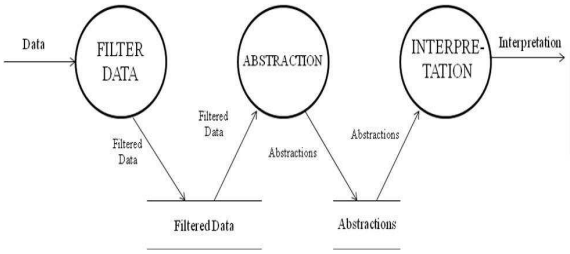

Figure 2: Data Flow Diagram of the INTERPRETOR System

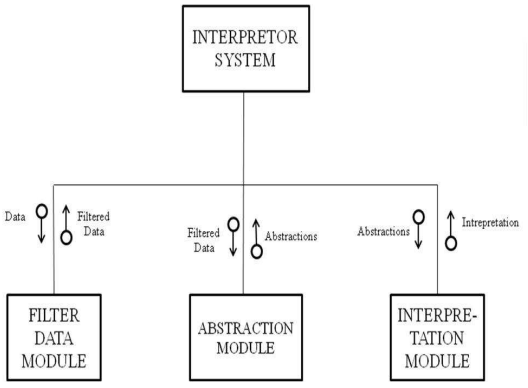

Figure 3: Overall Software Architecture of the INTERPRETOR System

It can be seen that INTERPRETOR is a data flow architecture. The architecture is decomposed into 3 processes which can be changed or replaced independently of the others - this makes INTERPRETOR a loosely coupled system. Indeed, each process of the INTERPRETOR performs one task or achieves a single objective - this makes the INTERPRETOR a highly cohesive system.

INTERPRETOR can be considered a pipe and filter architectural style because it provides a structure for systems that process a stream of data. Each processing step is encapsulated in a filter component (process) which reads streams of data on input and produces streams of data on output. A local incremental transformation to input stream is made and the output begins before input is consumed. Data is passed through pipes between adjacent filters - they are the conduits for streams and transmit outputs from one filter to input of another. The advantage of this approach is the overall behavior is a simple composition of behavior of individual filters. The architecture facilitates reuse so any two filters can be connected if they agree on that data format that is transmitted. There is ease of maintenance because any filter can be changed or replaced depending on the application.

We hope to extend our INTERPRETOR design architecture, such that we have a generic design pattern for voluminous and high frequency noisy data, whereby, the data is passed through 3 consecutive processes: *Filter Data* which takes the original data and removes outliers, inconsistencies or noise; *Abstraction* which takes the filtered data and abstracts features from the filtered data; and *Interpretation* which uses the abstractions and generates an interpretation of the original data.

4 Applications of INTERPRETOR

We will demonstrate the application of the INTERPRETOR software architecture to 2 case studies: Interpreting Intensive Care Unit (ICU) monitor data and interpreting building monitor data.

4.1 Case Study 1 - Interpreting ICU Monitor Data

The ICU bedside monitors confront the medical staff with large amounts of continuous noisy data - this is emphasised when there are many cardiovascular parameters such as the heart rate and blood pressure being recorded simultaneously. The frequency of the data can be higher than 1 value every second which creates information overload for medical staff who need to interpret the data to evaluate the state of the patient.

A system called ASSOCIATE [15] has been developed using the the INTERPRETOR software architecture to interpret the ICU monitor data. We shall describe how ASSOCIATE implemented each of the modules of the the INTERPRETOR software architecture.

4.1.1 Filter Module

1. copy the first k values of the physiological data to be the first k values of the cleaned physiological data
2. **for** n = (k+1) to (number of points in the physiological data– k) **do**
3. create a window of points from (n-k) to (n+k).
4. sort the values in this window – this will force extreme values to the ends of the window
5. set the n^{th} value of the cleaned physiological data to be the median value of the sorted window
6. **end for**
7. copy the last k values of the physiological data to be the last k values of the cleaned physiological data

Figure 4: Algorithm for Filter Data Module

Filtering is the process of removing certain noise like clinically insignificant events from the physiological parameters. Clinically insignificant events which can not be removed at this stage will be dealt with by the Interpretation process.

After various investigations of filtering techniques, a median filter was chosen. The median filter involves a moving window which is centered on a point xn and if the window is of size 2k+1 the window contains the points x_{n-k} to x_{n+k}. By always choosing the median value in the window as the filtered value, it will remove transient features lasting shorter than k without distortion of the base line signal; features lasting more than that will remain. A summary of the algorithm for applying the median filter to our physiological data is shown in Figure 4.

4.1.2 Abstraction Module

Given continuous data (up to one value every second), it is computationally expensive to reason with each data value on a point to point basis - this data needs to be reduced by performing abstraction. Abstraction is the classification of filtered data generated by the filtering process into temporal intervals (trends) in which data is steady, increasing and decreasing. One is also interested in the rate of change e.g rapidly increasing, slowly decreasing etc. One must decide the beginning and end of an interval.

Our algorithm for identifying trends involves following two consecutive sub-processes called temporal interpolation and temporal inferencing. Temporal interpolation takes the cleaned data and generates simple intervals between consecutive data point. Temporal inferencing takes these simple intervals and tries to generate trends - this is achieved using 4 variables: *diff* which is the variance allowed to derive steady trends, g_1 and g_2 which are gradient values used to derive increasing and decreasing trends and *dur* which is used to merge 3 intervals based on the duration of the middle interval. Temporal Inferencing rules to merge 2 meeting intervals (Δ_{H2}) and 3 meeting intervals (Δ_{H3}) use the 4 variables to try to merge intervals into larger intervals until no more merging can take place. The algorithm for abstraction is summarised in 5. For further discussion of the algorithm the reader is advised to read [17].

1. Apply the inferences in Δ_{H2} which derive only increasing or decreasing trends by trying to combine the first two intervals; if this succeeds try to combine this new interval with the next and so on. If combination fails, then we take the interval which failed to be combined, and use it as a new starting point.
2. Apply inferences in Δ_{H2} which derive only steady trends.
3. Set flag *still-to-do* to **true**.
4. **while** *still-to-do* **do**
5. Set *previous* to the number of intervals generated so far.
6. Apply the inferences in Δ_{H3}
7. Apply the inferences in Δ_{H2}
8. Set *still-to-do* to *previous* = current number of intervals.
9. **endwhile**

Figure 5: Algorithm for Abstraction Module

4.1.3 Interpretation Module

Interpretation is based on defining a trend template for each type of event we wish to identify - examples of trend templates are shown in Figure 6. A given trend template will specify criteria which apply both within intervals and between intervals. The two relationships of interest between intervals are: meeting where the end time of one interval is the same as the start time of the other; and overlapping where there exists a time which is common to both intervals.

The algorithm for interpretation involves applying the templates to the temporal intervals. Clinically insignificant event and clinical condition templates initially have the status absent and therapy templates initially have the status working. The reasoning engine assesses the status of the templates (i.e hypothesised or confirmed) by evaluating the expressions located in the HypothesiseConditions and ConfirmConditions slots with the data. Actions to be performed when the templates are hypothesised or confirmed are provided in the HypothesiseActions and ConfirmActions slots. If

we have a template which has a hypothesised status over a number of adjacent segments which are subsequently confirmed then in retrospect we change these hypothesised states to confirmed. This is a way of confirming our initial beliefs. All segments with clinically significant templates that have confirmed states represent the interpretation.

Trend templates encompass three types of knowledge: temporal, differential and taxonomical. Temporal knowledge allows temporal reasoning; interval-based and point-based reasoning. Interval-based temporal reasoning is achieved using the still_developing and together functions. Given a clinical condition which is described in terms of overlapping intervals, the still_devel oping function operates on the uncertain period between the hypothesised state and the confirmed state of the clinical condition. Here the still_developing function is satisfied if there is the correct temporal progression from the hypothesised state to the confirmed state. Similarly the together function operates on overlapping temporal intervals which make up clinically insignificant events. Here the together function is satisfied if the overall changes in all the individual parameters that make up the event all share a common time interval. Though defined differently, the together and still_developing functions take into account the expected changes of the individual parameters that make up specific events do not occur at exactly the same time.

Point-based temporal reasoning is used to determine the outcome of therapy. It is known that clinicians expect changes in parameters to be achieved by a lower and upper temporal bound represented as time points in the future. ASSOCIATE expresses point based temporal reasoning within temporal intervals. When therapy is administered at a specific point in time, we compare a (future) interval which contains the therapy's temporal bound (lower and upper) with the interval which contained the time of administration. We are interested in whether parameters have increased, decreased or remained the same in the future after the time of administration.

Since several clinical conditions may be described by the same patterns, differential knowledge can be used to eliminate possibilities and hence prevent unnecessary reasoning. Information such as the patient record which contains the patient's history can be used as differential knowledge.

Also within the trend templates there is taxonomical knowledge - since several clinical conditions have similar attributes, this enables us to represent them as a hierarchy of classes and subclasses. Such a representation allows more abstract clinical conditions to be identified

```
Interpretation respiratory_problem
   Description ("general class of respiratory condition")
   Type_of (clinical_condition)
   Preconditions (NIL)
   HypothesiseConditions( AND (paO2,_,_,steady)
                              (paCO2,_,_,increasing))
   ConfirmConditions( AND (paO2,_,_,decreasing)
                          (paCO2,_,_,increasing))
   HypothesiseActions (ALARM_WARN)
   ConfirmActions (ALARM_TRUE)
end_template

Interpretation pneumothorax
    Description ("pneumothorax")
    Type_of (respiratory_problem)
    Preconditions (NIL)
    HypothesiseConditions(AND  (mean_bp,_,_,steady)
                               (heart_rate,_,_,increasing))
    ConfirmConditions (AND (mean_bp _,_,increasing)
                           (heart_rate _,_,increasing))
    HypothesiseActions (ALARM_WARN)
    ConfirmActions (ALARM_TRUE)
end_template

Interpretation digoxin
   Description ("digoxin")
   Type_of (therapy)
   Preconditions (digoxin)
   HypothesiseConditions (heart_rate increased 10)
   ConfirmConditions (heart_rate increased 20)
   HypothesiseActions (ALARM_ALERT)
   ConfirmActions (ALARM_TRUE)
end_template

Interpretation transcutenous_probe_off
  Description ("transcutenous probe coming off")
  Type_of (insignificant_event)
  Preconditions (NIL)
  HypothesiseConditions (NIL)
  ConfirmConditions(AND (meeting (paO2,_,>16,increasing)
                                 (paO2,>16,>16,steady)
                                 (paO2, _, >16, decreasing))
                        (meeting (paCO2,<3,_,decreasing)
                                 (paCO2,<3,<3,steady)
                                 (paCO2, <3, _, increasing)))
   HypothesiseActions (NIL)
   ConfirmActions (REMOVE)
end_template
```

Figure 6: Possible templates for clinical conditions, insignificant events and therapies

- if a specific instance of a clinical conditions cannot be identified then the more general class of clinical condition to which it belongs is more likely to describe the data. For further discussion of the algorithm the reader is advised to read [15].

4.1.4 Results

ASSOCIATE has been tested on three datasets from an adult ICU (here each set covers 24 hours and contains measurements of heart rate, mean blood pressure and central venous pressure) and six datasets from a neonatal ICU (here each set covers about 60 hours and contains measurements of heart rate, mean blood pressure, partial pressure of oxygen and partial pressure of carbon dioxide). The data sets were taken in 1995 as part of a research project and the results were validated by a consultant aneasthetist and a consultant neonatologist.

Overall, ASSOCIATE has a false-positive rate of 28.9% and a false-negative rate of 0.3% in identifying clinically insignificant events, a false-positive rate of 10.7% and a false-negative rate of 0.15% in identifying clinical conditions and a false-positive rate of 0% and a false-negative rate of 87.9% in determining the outcome of therapy. Since all have a true positive rate which is higher than its false positive rate, ASSOCIATE can be seen as a conservative system [9].

As an example, consider a three day data set taken from an ICU from from 00:01 on 22 April 1995 to 23:59 on 24 April 1995; the frequency of the signal is one data item per minute. No prior knowledge of events that occurred within this data set was known to the expert or the tester. Figure 7 depicts the physiological data from ICU patient monitors and Figure 8 depicts a graphical summary of the temporal intervals generated for each parameter by the Abstraction Module. Note that in the graphs HR represents the Heart Rate, BP represents the Blood Pressure, PO represents the Partial Pressure of Oxygen and TCO represents the Partial Pressure of Carbon Dioxide.

All clinically insignificant events were correctly identified and removed.

For the clinical condition interpretation, the expert agrees that ASSOCIATE identified all 11 episodes of respiratory problems in the data. Of 2 of these episodes, namely those identified from 11:44 on 23/04/95 to 12:04 on 23/04/95 and from 13:57 on 23/04/95 to 14:32 on 23/04/95 may have been pneumothoraxes. However, ASSOCIATE incorrectly identifies respiratory problems on 5 occasions. ASSOCIATE also incorrectly identifies a pulmonary haemorrhage and a pneumothorax at the same time, though the expert agrees that there is a respiratory problem at this time.

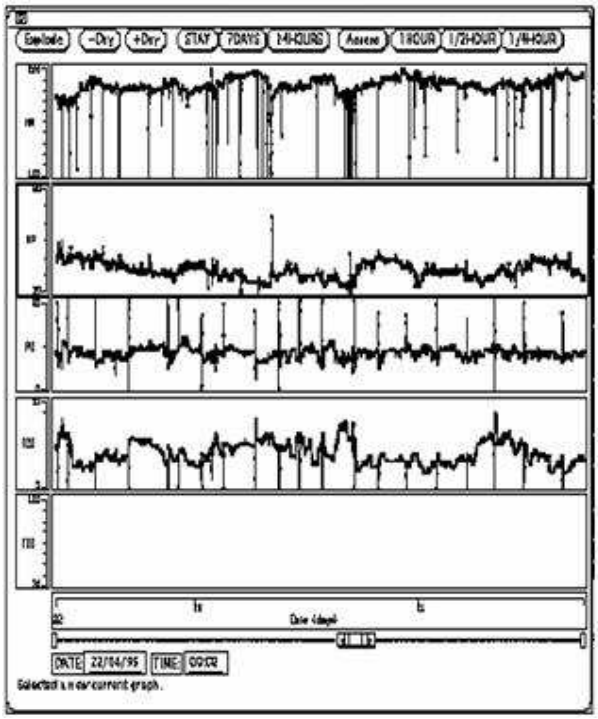

Figure 7: Original Physiological Data from ICU Patient Monitor

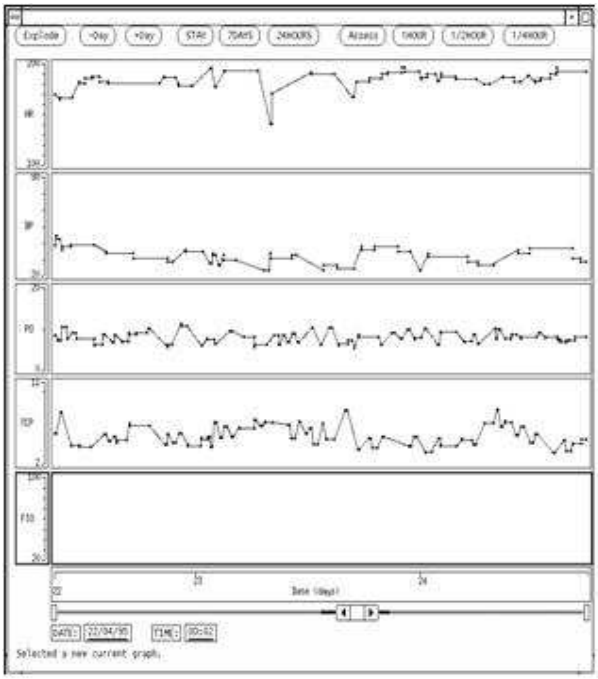

Figure 8: Graphical Summary generated by the Abstraction Module

ASSOCIATE identified 3 separate episodes of shock of which the expert agreed with 2 of them. The expert also agrees in ASSOCIATE's identifications of episodes of tachycardia and hypercarbia. However, a few of the episodes of hypoxaemia were incorrectly identified due to noisy data.

The expert agreed that ASSOCIATE recognised all clinical conditions in the data set i.e no clinical conditions were missed.

For the therapy interpretation, 6 therapies were administered. Of the 5 that worked ASSOCIATE correctly identifies 2 of them as working. ASSOCIATE correctly identifies the therapy that did not work. The incorrect results were because of noisy data and approximate times of administration.

4.2 Case Study 2 - Interpreting Building Sensor Data

Building operators are confronted with large volumes of continuous data from multiple environmental sensors which require interpretation. The ABSTRACTOR ([18], [16]) system used the INTERPRETOR software architecture to summarise historical building sensor data for interpretation and building performance assessment. We shall describe how ABSTRACTOR implemented each of the modules of the INTERPRETOR software architecture.

4.2.1 Filter Module

1. copy the first k values of the environmental data to be the first k values of the filtered environmental data
2. **for** n = (k+1) to (number of points in the environmental data- k) **do**
3. create a window of points from (n-k) to (n+k).
4. calculate the average of the values in this window
5. set the nth value of the filtered environmental data to be the average value of the window
6. **end for**
7. copy the last k values of the environmental data to be the last k values of the filtered environmental data

Figure 9: Algorithm for Filter Data Module

Initially data needs to be filtered to get rid of non-significant events in environmental monitoring data. Due to the nature and frequency of the data, an average filter was chosen - here all the very short duration spikes from the outdoor temperature data were removed whilst revealing the short duration trends hidden in the raw data. The algorithm for the filter module is given in Figure 9.

4.2.2 Abstraction Module

This module is exactly the same as the agglomerative approach used for case study 1 - for a discussion of this algorithm applied to building monitor data the reader is advised to read [20].

4.2.3 Interpretation Module

Given overlapping trends it is proposed, in the spirit of [6] they are split into global segments. A change in the direction of change (slope) of one (or more) channels or a change in the rate of change of one (or more) channels contributes to a split in the trends creating a global segment. A global segment can be considered as being a set of intervals - one for each channel.

```
if heat-flux increasing                 if heat-flux decreasing
   and ti-t0 decreasing then               and ti-t0 steady then
         fault detected                          fault detected
end if                                  end if

if heat-flux increasing                 if heat-flux steady
   and ti-t0 steady then                   and ti-t0 increasing then
         fault detected                          fault detected
end if                                  end if

if heat-flux decreasing                 if heat-flux steady
   and ti-t0 increasing then               and ti-t0 decreasing then
         fault detected                          fault detected
end if                                  end if
```

Figure 10: Example of rules to apply to global segments

The algorithm for interpretation involves applying rules to the global segments. Examples of rules for identifying faults are shown in Figure 10 - here a fault is declared when the heat-flux does not have the same trend as the difference in internal and external temperature (t1-t0). If rules are true over adjacent global segments then one can determine when the fault started and ended.

4.2.4 Results

ABSTRACTOR has been tested on over 8 days (121 79 minutes) worth of continuous data (see Figure 11a). The data was the heat-flux into a wall and the difference in internal and external temperature (ti-t0) measurements; the sampling frequency of the signals is one data item every 15 minutes. No prior knowledge of events that occurred within this data set was known to the expert or the tester. The application of the average filter (k=10 filter provides a running five and a quarter hour running average) is shown in the middle graph 11(b) and the intervals generated are shown in the bottom graph 11(c).

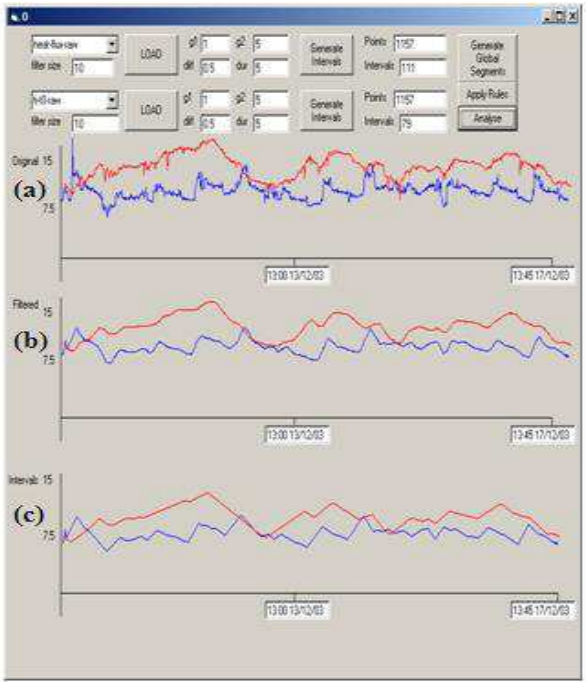

Figure 11: Output of ABSTRACTOR

Overall, ABSTRACTOR has a sensitivity of 56%, specificity of 64%, predictive value of 43%, a false positive rate of 57% and a false negative rate of 24%. These results mean that when a fault is present ABSTRACTOR is detecting it only 56% of the time but when there is no fault it will correctly identify this 64% of the time. Whilst it would seem that ABSTRACTOR is only slightly better than tossing a coin to decide the presence or absence of a fault it needs to be remembered that the actual fault conditions were derived from an expert's manual abstraction of the raw data which is dependent on the expert's attitude and experience. A direct comparison with the raw data is meaningless because the data is at intervals much shorter than the trends. If ABSTRACTOR were to be incorporated in its present state into a control system it would generate a high number of false alarms (57%) but would fail to detect a fault only 24% of the time. These results are indicating that ABSTRACTOR is a more liberal system than a random system [9].

5 Conclusions

The interpretation of high frequency and noisy data is non-trivial. The INTERPRETOR software architecture is designed in such a way to allow the interpretation of high frequency and noisy data and the results of INTERPRETOR are encouraging. We have shown that INTERPRETOR can be applied to different domains which have the same issues associated with the interpretation of voluminous and noisy data. Our future work will be to develop a tool for our software architecture which should lend itself for reuse and then validate it with further case studies.

In summary, INTERPRETOR reasons with multiple signals in an intuitive way. Although it is not perfect, it is a step forward in the development of systems for the interpretation of voluminous high frequency and noisy data.

References

[1] Beigl, M., Beuster, M., Rohr, D., Riedel, T., Decker, C., and Krohn, A. S2b2: Blackboard for transparent data and control access in heterogeneous sensing systems. pages 126–129, 2007.

[2] Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. Jade: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, 50(1-2):10–21, 2008.

[3] Carrascosa, C., Bajo, J., Julian, V., Corchado, J. M., and Botti, V. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34(1):2–17, 2008.

[4] Crawford, C. H., Bate, G. P., Cherbakov, L., Holley, K., and Tsocanos, C. Toward an on demand service-oriented architecture. *IBM Systems Journal*, 44(1):81–107, 2005.

[5] Dawant, B. M., Uckun, S., Manders, E. J., and Lindstrom, D. P. Soda: Service oriented device architecturethe simon project - model-based signal acquisition, analysis, and interpretation in intelligent patient monitoring. *IEEE Engineering In Medicine and Biology*, 12(4):82–91, 1993.

[6] DeCoste, D. Dynamic across-time measurement interpretation. *Artificial Intelligence*, 51:273–341, 1991.

[7] Decruyenaere, J., DeTurck, F., Vanhastel, S., Vandermeulen, F., P Demeester, P., and deMoor, G. On the design of a generic and scalable multilayer software architecture for data flow management in the intensive care unit. *Journal of Methods of Information in Medicine*, 3:79–88, 2010.

[8] deDeugd, S., Carroll, R., Kelly, K. E., Millett, B., and Ricker, J. Soda: Service oriented device architecture. *IEEE Pervasive Computing*, 5(3):94–96, 2006.

[9] Fawcett, T. Roc graphs: Notes and practical considerations for data mining researchers. Technical Report HPL-2003-4, Intelligent Enterprise Technologies Laboratory, HP Labs Palo Alto, January 2003.

[10] Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., and Seiver, A. Guardian: A prototype intelligent agent for intensive-care monitoring. Technical Report KSL 91-42, Knowledge Systems Lab Report, Department of Computer Science, Stanford University, June 1991.

[11] Miksch, S., Horn, W., Popow, C., and Paky, F. Context-sensitive data validation and data abstraction for knowledge-based monitoring. Technical Report TR-94-04, Austrian Research Institute for Artificial Intelligence, 1994.

[12] Mistry, M. and Shah, D. Implementation of multi-agents system in health care domain. pages 100–104, January 2011.

[13] Rich, E. *Artificial Intelligence*. 1988.

[14] Rudenko, D. and Borisov, A. An overview of blackboard architecture application for real tasks. volume 31, pages 50–57, 2007.

[15] Salatian, A. Interpreting historical icu data using associational and temporal reasoning. volume 4, pages 442–450, 2003.

[16] Salatian, A. A software architecture for decision support of building sensor data. *International Journal of Smart Home*, 4(4):27–34, 2010.

[17] Salatian, A. and Hunter, J. R. W. Deriving trends in historical and real-time continuously sampled medical data. *Journal of Intelligent Information Systems*, 13:47–74, 1999.

[18] Salatian, A. and Oriogun, P. A software architecture for summarising and interpreting icu monitor data. *International Journal of Software Engineering*, 4(1):3–14, 2011.

[19] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. New Jersey, 1996.

[20] VanHoecke, S., DeTurck, F., Danneels, C., DeProft, K., Taveirne, K., and Decruyenaere, J. Platform for intelligent agent subscription in icu. 2006.

[21] Yang, M., Wang, S., Abdelal, A., Jiang, Y., and Kim, Y. An improved multi-layered architecture and its rotational scheme for large-scale wireless sensor networks. *Las Vegas, NV, USA*, pages 855–859, Janaury 2007.

Universidade Federal de Lavras
INFOCOMP Journal of Computer Science
Publication guidelines (July of 2011)

1. INFOCOMP publishes original scientific and technological papers in English. The papers should be related to Computer Science.

2. The papers should be submitted in PDF format without authors' informations using the JEMS system (https://submissoes.sbc.org.br/infocomp) of the Brazilian Computer Society (SBC). Login and password are necessary and they can be obtained online in the JEMS system.

3. The papers should follow the INFOCOMP format: Paper Letter (21.5x28.0cm), 1.1 line spacing, Times New Roman 10, justified text, with superior, inferior, left and right margins of 2.5 cm. The number of pages should not exceed 12. Papers with more than 12 pages may be accepted after analysis by the editorial board.

4. The first page should contain title, authors (only in the final version), abstract and keywords. Afterwards, it should contain the following text centralized: "(Received January 1st, 2005 / Accepted December 31st, 2005)" for posterior edition with the correct dates. These informations should not exceed one page and should be in one column. The title should be in font size 14.

5. The authors' names (only in the final and accepted version) should be sided horizontally, identified by a superscripted number. The affiliation and the electronic address of each author should be written right below the names.

6. The text of the paper should be formatted in two columns, separated by 0.5cm, with numerated sections and subsections. Examples are: 1. Introduction, 1.1. Terminology. Figures and tables may occupy the page width, if necessary. The figure and table titles should be numerated and centralized below them.

7. The references should be numerated and listed in alphabetical order. Ex: [5] Hougardy, S. Even pairs and the strong perfect graph conjecture, Discrete Math. v.154, p.277-288, 1996. Citations should be made based on the number of the reference. "In [5], it was proved that..." is an example of citation.

8. Footnotes may be accepted, when strongly necessary, for explanations that cannot be included in the text, such as: (a) name of the research institution; (b) support organizations and other financial supports; (c) reference to the publication as part of MSc or PhD thesis; (d) personal communication.

9. The authors give the right of publishing and formatting the articles upon submission.

10. Non compliance with these rules will result in the non acceptance of the paper.

11. Publications in the INFOCOMP are free of charge.