

Managing Failures in a Grid System using FailRank*

Demetrios Zeinalipour-Yazti, Kyriacos Neocleous,
Chryssis Georgiou, Marios D. Dikaiakos

Department of Computer Science, University of Cyprus
{dzeina,kyriacos,chryssis,mdd}@cs.ucy.ac.cy

Abstract

The objective of Grid computing is to make processing power as accessible and easy to use as electricity and water. The last decade has seen an unprecedented growth in Grid infrastructures which nowadays enables large-scale deployment of applications in the scientific computation domain. One of the main challenges in realizing the full potential of Grids, is to make these systems *dependable*.

The objective of this paper is threefold: Firstly, we would like to acquaint the dependability community with the challenges of realizing a dependable Grid. Secondly, we identify the causes of failures in Grid jobs. Our findings are extrapolated from the experiences we acquired by operating a 72 CPU site of the EGEE Grid, one of the largest Grids in scientific computing, for a time span of two years. Lastly, we describe the *FailRank* architecture, a novel framework for integrating and ranking information sources that characterize failures in a grid system. We identify challenges and preliminary solutions for a variety of complementary tasks including exploratory data analysis and prediction.

Technical Report TR-06-4
Department of Computer Science
University of Cyprus
September, 2006

*This work was supported in part by the European Union under projects EGEE (#IST-2003-508833) and CoreGRID (#IST-2002-004265).

1 Introduction

Grids have emerged as wide-scale, distributed infrastructures that comprise heterogeneous computing and storage resources, operating over open standards and distributed administration control [11, 12]. Grids are quickly gaining popularity, especially in the scientific sector, where projects like *EGEE (Enabling Grids for E-science)* [7], TeraGrid [24], the \$53M project by NSF, and *Open Science Grid* [22], provide the infrastructure that accommodates large experiments with thousands of scientists, tens of thousands of computers, trillions of commands per second and petabytes of storage [7, 24, 22]. At the time of this writing, EGEE assembles over 180 sites around the world with more than 30,000 CPUs and about 5PB of storage, supporting over 80 Virtual Organizations.¹

While the aforementioned discussion shows that Grid Computing will play a vital role in many different scientific domains, realizing its full potential will require to make these infrastructures *dependable*. As a measure of dependability of grids we use the ratio of successfully fulfilled job requests over the total number of jobs submitted to a grid infrastructure. The WISDOM [28] project and our findings presented in Section 2, indicate that only a 65% of the user submitted jobs are executed successfully. Consequently, the dependability of large-scale grids needs to be improved substantially.

Detecting and managing failures is an important step toward the goal of a dependable grid. Currently, this is an extremely complex task that relies on over-provisioning of resources, ad-hoc monitoring and user intervention. Adapting ideas from other contexts such as cluster computing [19], Internet services [16, 17] and software systems [20] seems also difficult due to intrinsic characteristics of a grid environment. First, note that a grid system is not administered centrally, thus it is hard to access the remote sites in order to monitor failures. Secondly, these systems are extremely large; thus, it is difficult to acquire and analyze failure feedback. Lastly and more importantly, in a grid environment it is more important to identify quickly the overall state of the system, so that we can exclude failing sites from the job scheduling process, rather than identifying many individual failure feedbacks. Of course the latter information will be essential to identify the root cause of a failure [17], but this operation is usually performed in some offline phase that is complementary to our work.

In the FailRank architecture, feedback sources (e.g. websites, LDAP queries, representative low-level measurements, etc) are continuously coalesced into a representative array of numeric vectors, the *FailShot Matrix (FSM)*. FSM is then continuously ranked in order to identify the K sites with the highest potential to feature some failure. This allows the system to automatically exclude the respective sites from the job scheduling process. Our prototype system, currently under development, also enables other complex knowledge extrapolation tasks that are utilized for learning and prediction.

The advantages of our approach can be summarized as following: (i) FailRank is a simple yet powerful framework to integrate and quantify the multi-dimensional parameters that affect failures in a grid system; (ii) Our system is tunable, allowing system administrators to drive the ranking process through user defined ranking functions; (iii) We eliminated the need for human intervention, thus our approach gives space for automated exploitation of the extracted failure semantics; (iv) We expect that the FailRank logic will be implemented inside the Grid job scheduler (Resource Broker), thus imposing minimum changes to a Grid infrastructure.

2 Background on Grid Computing

In this section we will describe the anatomy of a Grid system and detail all the components pertinent to the operation of a Grid site. In particular, we will focus on Grid computing in the context of the EGEE project although other architectures feature a similar framework. We also describe the main causes of unsuccessful job executions in a grid system as well as popular failure feedback mechanisms.

¹A *Virtual Organization (VO)* can be thought as a federation between entities that share a common objective (e.g. Institutions with in an interest in the High Energy Physics put their computational resources together).

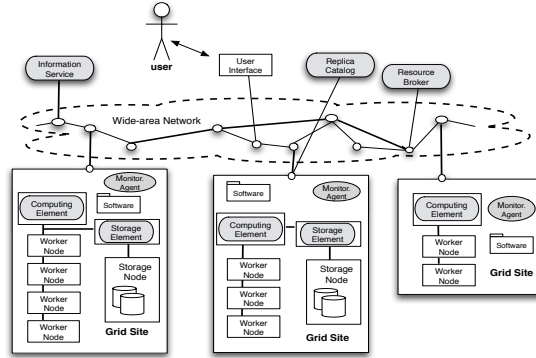


Figure 1: The Anatomy of a GRID Infrastructure.

2.1 The Anatomy of a Grid

A Grid interconnects a number of remote clusters, or *sites*. Each site features heterogeneous resources (hardware and software) and the sites are interconnected over an open network such as the Internet. Figure 1 illustrates the anatomy of a typical grid (rectangles represent hardware while ellipses the services). The figure shows how sites with different capabilities and capacities are contributing their resources to the Grid infrastructure. In particular, each site features one or more *Worker Nodes*, which are usually rack-mounted PCs. The *Computing Element* shown in the same figure runs various services responsible for authenticating users, accepting jobs, performing resource management and job scheduling. Additionally, each site might feature a *Local Storage* site, on which temporary computation results can reside, and local *Software* libraries, that can be utilized by executing processes. The Grid middleware is the component that glues together local resources and services and exposes high-level programming and communication functionalities to application programmers and end-users. For instance EGEE uses the LCG middleware [18], while NSF's TeraGrid is based on the Globus Toolkit [13]. A Grid system also features some global services which are described in the next subsection.

2.2 A Lifecycle of Grid Jobs

A Grid job, or computation, consists of a set of input files that defines the elements of a given computation (code, custom libraries, input files, etc). Grid jobs can be classified as *CPU-intensive* and *data-intensive*, depending on the type of work performed. For clarity we divide the lifecycle of a grid job into the following three conceptual phases:

i) Assignment Phase: Jobs are submitted to a Grid by users through some authenticated remote workstation, denoted as the *User Interface (UI)*. Besides obtaining the output from completed jobs, the UI might also provide supplementary functionality for requesting the status of a job and the status of resources in the system. Jobs submitted to the UI are directed to some *Resource Broker (RB)*, a central global grid service that performs matching between requests and available resources using the *matchmaking* approach [23]. Being able to quickly identify failures, would obviously be very helpful information to the *RB* as it would be able to avoid bottlenecks and resources leading to errors. Although this is not currently possible, our work sets the foundation towards this goal. The matchmaking performed by the *RB* is based on the information provided by another central service, the *Information Index*, which provides information about the state of grid resources. If the matchmaking is successful, the job is sent to the respective computing elements for execution.

ii) Execution Phase: During job execution, if any *input files* are necessary, these have to be pushed to a remote grid site at runtime. Alternatively these files could have been pushed to the grid site during the assignment phase. In both occasions, a service called the *Replica Catalog* maintains the location of various replicas of a file held in remote Storage Elements.

ii) Completion Phase: When the job completes successfully, the user is informed through the User Interface with a set of output files that are a superset of the command line outputs, had the job run on a standalone computer. Although the user will be notified in the event of a failure, there is no indication about the possible cause.

2.3 Causes of Failures

In this section we identify the main causes of failures in Grid infrastructures. These observations are extrapolated from the experiences we acquired by operating an EGEE grid site that consists of: (i) a Regional Resource Broker (3.6GHz/1GB RAM), (ii) a Regional Information Service which features the same aforementioned characteristics, (iii) a 72 CPU cluster of Worker Nodes which utilizes a blend of 2.6GHz AMD Opteron and 2.8GHz Xeon CPUs, and (iv) a Storage Element which features 4x250GB disk space in RAID 5. Our analysis takes into account 37,860 job submissions ($\approx 19\text{K}$ normalized CPU hours), between March 2005 and June 2006. We combine our observations with others obtained by fellow-researchers [28, 8, 15] to conclude the following:

Grid component failures: One or more of the components involved in the Grid infrastructure could malfunction due to *hardware failures* (e.g., hard drive burns, RAM or motherboard failures, power supply failures and overheating) and *software faults* (e.g., O/S misconfigurations and middleware bugs). Such problems may result to a total collapse of a component (crash failure) or to a component becoming partially unresponsive or extremely slow. The combined studies suggest that 25 – 30% of the incomplete jobs are due to Grid component failures.

Network failures: Network links could cause permanent or transient network disconnections leading to a loss, corruption or delay of messages and data transfers. Network disconnections may result to total inaccessibility of a Grid component, a condition that is equivalent to a crash failure of that component. Network access misconfiguration (firewall changes or updates) lead to the same effect. Our findings suggest that 10 – 15% of the incompleting jobs are due to Network failures.

Information faults: The information provided by the *Grid Information Service*, which provides state information about the distributed grid sites, may be erroneous or obsolete due to administrator errors, software faults, and network delays. As a result, the *Resource Broker*, a central service that performs matching between resources and requests based on this information, may take suboptimal decisions that result to excessive delays in job processing or even to failures in job execution. Our findings indicate that 30% of the incompleting jobs are due to such faults.

Excessive delays: In the large, shared and dynamic Grid infrastructure, unusual workload conditions, like those triggered by flash crowds and denial of service attacks, may lead to long queuing delays in Computing or Storage Elements, to reduced Grid service throughput, and to long network delays in data transfers. Such conditions may result to job turnaround times that are substantially longer than those expected by Grid users. A similar effect may arise also because of the heterogeneity of the Grid: jobs may end-up being executed on very slow resources, resulting to unacceptably slow execution times. Because of the resource virtualization imposed by many Grids, end-users have limited control over the performance characteristics of resources allocated to their jobs. Although a clear percentage attributing to excessive delays cannot be extracted, as delays are also caused by network delays or information faults, we estimate that 20% of incompleting jobs are due to such faults.

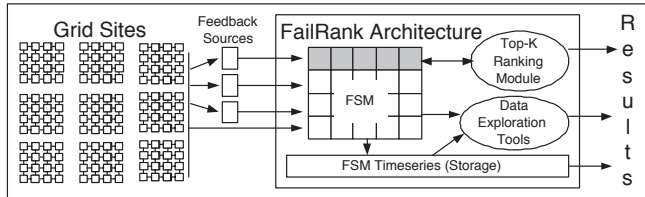


Figure 2: The FailRank System Architecture.

2.4 Failure Feedback Sources

In this subsection we overview typical failure feedback sources provided in a Grid environment. Our discussion is again in the context of the EGEE infrastructure, but similar tools and sources exist in other grids [24, 22].

Meta-Information Sources: Several methods for detecting failures have been deployed so far. Examples include: (i) *Site Functional Test (SFT)* report website [29]: a reporting web site is maintained for publishing periodic test-job results for all sites of the infrastructure. (ii) *Global Grid User Support (GGUS)* ticketing system [9]: this system is used to report component failures as well as needed updates for sites. Such tickets are typically opened due to errors appeared in the SFT reports. (iii) *Core Infrastructure Center (CIC)* broadcasts [5]: site managers report site downtime events to all affected parties through the CIC website. (iv) *LDAP Queries*: queries that can be performed on the Information Service to extract fine-grained information regarding the complete status of a grid site. (v) *Machine log-file investigation*: error information is extracted by the grid nodes' log files (that are automatically maintained by each node).

All the above methods have a major drawback: their success relies heavily on human (administrator) intervention. As Grid infrastructures become larger, human intervention becomes less feasible and efficient. As we would like Grid Dependability to be scalable, our proposed architecture does not rely on human intervention but provides *automated* means for achieving it.

Active benchmarking: Deploying a number of lower level probes to the remote sites is another direction towards the extraction of meaningful failure semantics. In particular, one can utilize tools such as GridBench [25, 26], the Grid Assessment Probes [4] and DiPerF [6], in order to determine, in real time, the value of certain low level and application level failure semantics that can not be furnished by the meta-information sources. For example, GridBench provides a corpus of over 20 benchmarks while the LLCbench suite [21], enables fine-grained performance measurements about the memory subsystem or the performance of some critical MPI functions. In addition, one could actively *ping* and *traceroute* other hosts, in order to obtain delay, loss and topological structure information about the network. The main drawbacks of these approaches is that they introduce some additional network and processing overhead and that they mandate the existence of special access privileges (VO memberships). Nevertheless, these approaches enable the extraction of fine-grained failure semantics which can complement the information base collected and maintained by the FailRank architecture.

3 The FailRank System

In this section we describe the underlying structure that supports the FailRank system. We start out with an architecture overview and then proceed with basic definitions in order to formalize our description. We follow with the description of the failure ranking mechanism deployed in FailRank. The section concludes with an outline of how our architecture provides the foundation for supporting a wide array of failure-extrapolation tasks that have not been addressed in the context of a Grid.

Site	CPU	DISK	QUEUE	NET	FAIL
s_1 ="USC-LCG2"	0.63	0.61	0.01	0.28	0.35
s_2 ="TAU-LCG2"	0.66	0.91	0.92	0.56	0.58
s_3 ="ELTE"	0.48	0.01	0.16	0.56	0.54
s_4 ="UCL-CCC"	0.99	0.90	0.75	0.74	0.67
s_5 ="CY01-LCG2"	0.44	0.07	0.70	0.19	0.67

Table 1: The *FailShot Matrix (FSM)* coalesces the failure information, available in a variety of formats and sources, into a representative array of numeric vectors.

3.1 Architecture Overview

The FailRank architecture (see Figure 2), consists of four major components: (i) A *FailShot Matrix (FSM)*, which is a compact representation of the parameters that contribute to failures, as these are furnished by the feedback sources; (ii) A temporal sequence of FSMs defines an *FSM timeseries* which is stored on local disk; (iii) A *Top-k Ranking Module* which continuously ranks the FSM matrix and identifies the K sites with the highest potential to run into a failure using a user defined scoring function; and (iv) A set of data exploration tools which allow the extraction of failure trends, similarities, enable learning and prediction. FailRank is tunable because it allows system administrators and domain experts to drive the ranking process through the provisioning of custom scoring functions.

3.2 Definitions and System Model

Definition (*FailShot Matrix (FSM)*): Let S denote a set of n grid sites (i.e. $S = \{s_1, s_2, \dots, s_n\}$). Also assume that each element in S is characterized by a set of m attributes (i.e. $A = \{a_1, a_2, \dots, a_m\}$). These attributes are generated by the feedback sources described in Section 2.4. The rows in Table 1, represent the sites while the columns represent the respective attributes.² The j -th attribute specifies a *rating* (or *score*) which characterizes some grid site s_i ($i \leq n$) at a given time moment. These ratings are extracted by custom-implemented parsers, which map the respective information to real numerics in the range $[0..1]$ (1 denotes a higher value towards failure). The $m \times n$ table of scores defines the *FailShot Matrix (FSM)*, while a *Site Vector* is any of the n rows of FSM.

A graphical illustration for some synthetic example is given in Table 1. The figure shows five sites $\{s_1, \dots, s_5\}$ where each site is characterized by five attributes: CPU (% of cpu units utilized), DISK (% of storage occupied), QUEUE (% of job queue occupied), NET (% of dropped network packets) and FAIL (% of jobs that don't complete with an "OK" status).

Definition (*FSM Timeseries*): A temporal sequence of l FailShot Matrices defines an *FSM Timeseries of order l* .

Keeping a history of the failure state for various prior time instances is important as it will enable us to automatically post-analyze the dimensions that contributed to some failure, learn the system behavior, predict failures etc. It is important to notice that the FSM timeseries can be stored incrementally in order to reduce the amount of storage required to keep the matrix on disk. Nevertheless, even the naive storage plan of storing each FSM in its entirety, is still orders of magnitudes more storage efficient than keeping the raw html/text sources provided by the feedback sources.

3.3 The Ranking Module

Although the snapshot of site vectors in FSM greatly simplifies the representation of information coming from different sources, observing individually hundreds of parameters in real time is still a difficult task.

²The j^{th} attribute of the i^{th} site is denoted as s_{ij} .

CPU	DISK	QUEUE	NET	FAIL	RANK
$s_4, .99$	$s_2, .91$	$s_2, .92$	$s_4, .74$	$s_4, .67$	$s_4, 4.05$
$s_2, .66$	$s_4, .90$	$s_4, .75$	$s_2, .56$	$s_5, .67$	$s_2, 3.63$
$s_1, .63$	$s_1, .61$	$s_5, .70$	$s_3, .56$	$s_2, .58$	$s_5, 2.07$
$s_3, .48$	$s_5, .07$	$s_3, .16$	$s_1, .28$	$s_3, .54$	$s_1, 1.88$
$s_5, .44$	$s_3, .01$	$s_1, .01$	$s_5, .19$	$s_1, .35$	$s_3, 1.75$

Table 2: The Sorted (by column score) FSM (Sorted-FSM) is utilized by the top-k engine to continuously identify K highest ranked answers, where K is a user parameter.

For example a typical LDAP query to the Grid Information Service offers around 200 attributes. Instead, inferring a more general picture of the potential causes of failures in the system rather than observing individually parameters such as waiting jobs, cpu load, etc, is more meaningful. Ideally, one would like to provide, in some concise representation, the sites with the highest potential to suffer from failures. Since this information will be manipulated in high frequencies, we focus on computing the K sites with the highest potential to suffer from failures rather than finding all of them. Therefore we don't have to manipulate the whole universe of answers but only the K most important answers, quickly and efficiently. The answer will allow the Resource Broker to automatically and dynamically divert job submissions away from sites running into problems as well administrators to take preventive measures for the sites more prone to failures.

Scoring Function: In order to rank sites we utilize some aggregate scoring function which is provided by the user (or system administrator). For ease of exposition we use, similarly to [2, 30], the function:

$$Score(s_i) = \sum_{j=1}^m w_j * s_{ij} \quad (1)$$

where s_{ij} denotes the score for the j^{th} attribute of the i^{th} site and w_j ($w_j > 0$) a weight factor which calibrates the significance of each attribute according to the user preferences. For example if the CPU load is more significant than the DISK load, then the former parameter is given a higher weight. Shall we require to capture more complex interactions between different dimensions of FSM, we could construct, with the help of a domain expert, a custom scoring function or we could train such a function automatically using historic information. It is expected that the scoring function will be much more complex in a real setting (e.g. a linear combination of averages over n' correlated attributes, where $n' \ll n$) and thus minimizing its computation is an important factor.

Example: In order to stimulate our description, consider the example of Table 1. In order to infer the overall rank for two site vectors, such as $s_2 = \{0.66, 0.91, 0.92, 0.56, 0.58\}$ and $s_4 = \{0.99, 0.90, 0.75, 0.74, 0.67\}$, we apply the scoring function with $w_j = 1$ (i.e. all dimensions are of equal importance), and find that $s_2 = 3.63$ and $s_4 = 4.05$.

In order to minimize the computation of the scoring function, which potentially has to join hundreds of columns in each run, we utilize the *Threshold Algorithm (TA)* [10]. *TA* is one of the most widely recognized algorithms for finding the K highest rank answers in database and middleware scenarios. Suppose that we are interested in finding the $K = 1$ objects with the highest score. *TA* starts out by performing a parallel access to the n lists of the Sorted-FSM (see Table 2). While an object s_i is seen, *TA* performs a random access to the other lists to find the exact score for s_i using the given scoring function. In our working example the exact score would be computed for the two objects in the first row³ (i.e. $s_4 = 4.05$ and $s_2 = 3.63$). It then computes a *threshold* value τ as the sum of all scores in the first row (i.e. $\tau = .99 + .91 + .92 + .74 + .67 = 4.23$). Since τ is larger than both scores of s_4 and s_2 , the *TA* algorithm performs another iteration in which the threshold τ is refined as the sum of scores across the second row (i.e. $\tau = 3.54$). It also computes the exact

³Sorted access is executed on a row-at-a-time basis

score for $s_5 = 2.07$ (the only unresolved object in the second row). Now the algorithm finds at least $K=1$ objects above the threshold (i.e. $s_4 \geq \tau$ and $s_2 \geq \tau$) and therefore terminates. It is easy to prove that no other object (s_1 and s_3) can have a score above s_4 thus the score function calculation can be omitted for these objects.

3.4 Other Applications utilizing FailRank

In this subsection we review some exploratory data analysis, learning and prediction applications that are built on top of the FailRank architecture.

(i) Finding State-related Sites: An interesting question is whether any pair of sites features a similar *site vector*. This is an indication that two or more sites are in a similar failure state, with regards to the attributes of FSM. In order to answer this question we need a method that compares two vectors (\vec{q} , \vec{s}_i), and finds if these are similar. An efficient technique, widely used in the information retrieval domain, is the *cosine similarity* [14]. The cosine similarity finds the cosine of the angle between two vectors. If two vectors are identical then the cosine similarity is 1 (because the angle between them is 0). On the contrary if two vectors are different then the similarity is closer to 0.

The cosine similarity is calculated as following:

$$sim(q, s_i) = \frac{\sum(\vec{q} * \vec{s}_i)}{\sqrt{\sum(\vec{q})^2} * \sqrt{\sum(\vec{s}_i)^2}} \quad (2)$$

By executing the cosine similarity for the sites in Table 2, we find that the highest similarity is $sim(s_2, s_4) = 0.97$ while the smallest is $sim(s_1, s_5) = 0.57$.

(ii) Timeseries Similarity Search: Identifying which attribute timeseries are similar allows us to find the correlated attributes in FSM. For instance we can find that the QUEUE timeseries is correlated to the CPU timeseries for some site. To formalize our description, let $P = (p_1, p_2, \dots, p_l)$ and $Q = (q_1, q_2, \dots, q_l)$ denote two 1-dimensional timeseries of length l (each point denotes some item s_{ij} in FSM).

The most straightforward way to compute the similarity between P and Q is to apply the *Euclidean distance* (L_2) which is given by $d = |P - Q| = \sqrt{\sum_{i=1}^l |p_i - q_i|^2}$. Since data points are only matched at identical time positions, the running time of this approach is $O(l)$. However this distance is not able to handle *out-of-phase matches*. To understand this consider two identical timeseries P and Q , where Q is shifted in time by some offset t (i.e. $p_i = q_{i+t}, \forall i \leq l$). Using L_2 would obviously not yield any similarity between P and Q . The Dynamic Time Warping (DTW) [1], Longest Common Sub-Sequence (LCSS) [3] and the Upper LCSS method [27], utilized in our system, allow local stretching by matching each point of P with other points of Q within some window δ (i.e. p_i is matched with $q_{i \pm \delta}, \forall i \leq l$). This allows us to correlate noisy failure timeseries with out-of-phase matches again in an effective manner (in $O(l)$).

(iii) Decision Tree Learning: Given a site vector $s_i = \{a_1, \dots, a_m\}$, we want to predict if s_i will fail (with some statistical confidence). To answer this question, we train a Decision tree T [14] in an offline phase using a corpus of annotated failures. We then extract the classification rules that are utilized by the FailRank system. For instance if we learn that a site vector of the form $\{CPU \geq 0.70, DISK \geq 0.90, QUEUE \geq 0.85, \text{any}, \text{any}\}$ fails in 95% of the cases, then sites satisfying this rule are excluded from the job scheduling process. An interesting problem is to provide a decision tree which continues its learning behavior even after the initiation of the system and which gracefully adapts to changes.

4 Conclusions & Future Work

In this paper we introduce FailRank, a novel framework for integrating and ranking information sources that characterize failures in a grid system. FailRank supports a wide array of failure extrapolation tasks by

utilizing well established data exploratory analysis tools. In the future we plan to devise techniques to align temporal data sampled at different frequencies (e.g. one series is sampled every 1 minute while the other is sampled every 10 minutes). We also plan to develop storage and access-efficient data structures that will allow us to manipulate the FSM timeseries more efficiently. Finally, we plan to assess the performance of our prototype system, under development, that realizes many of the described ideas.

References

- [1] Berndt D. , Clifford J., “Using Dynamic Time Warping to Find Patterns in Time Series”, In *KDD* 1994.
- [2] Bruno N., Gravano L. and Marian A., “Evaluating Top-K Queries Over Web Accessible Databases”, In *ICDE* 2002.
- [3] Das G., Gunopulos D., Mannila H., “Finding Similar Time Series”, In *PKDD*, 1997.
- [4] Chun G., Dail H., Casanova H., and Snaveley A., “Benchmark probes for grid assessment”, In *IEEE IPDPS* 2004.
- [5] “CIC”, <http://cic.in2p3.fr/>
- [6] Dumitrescu C., Raicu I., Ripeanu M., Foster I., “DiPerF: An automated DIstributed PERformance testing Framework”, In *IEEE/ACM Grid* 2004.
- [7] “EGEE”, <http://www.eu-egee.org/>.
- [8] “EGEE JRA2: Quality assurance”, <http://egee-jra2.web.cern.ch/EGEE-JRA2/>
- [9] “Global Grid User Support (GGUS) ticketing”, <https://gus.fzk.de/pages/home.php>
- [10] Fagin R., Lotem A. and Naor M., “Optimal Aggregation Algorithms For Middleware”, In *PODS* 2001.
- [11] Foster I. and Kesselman C., “The Grid: Blueprint for a New Computing Infrastructure”, Elsevier, 2004.
- [12] Foster I., Kesselman C., and Tuecke S., “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, In *Intl. J. Supercomputer Applications*, 15(3):200–222, 2001.
- [13] Foster I., “Globus Toolkit Version 4: Software for Service-Oriented Systems”, In *IFIP ICNP* 2005.
- [14] Han J. Kamber M., “Data Mining: Concepts and Techniques”, 2nd edition, Elsevier, 2006.
- [15] Junqueira, F. P., and Marzullo, K., “The virtue of dependent failures in multi-site systems”, In *HotDep* 2005.
- [16] Kiciman E. and Fox A., “Detecting Application-Level Failures in Component-based Internet Services”, In *IEEE Transactions on Neural Networks*, 2004.
- [17] Kiciman E. and Subramanian L., “Root Cause Localization in Large Scale Systems”, In *HotDep* 2005.
- [18] “Large Hadron Collider Computing Grid (LCG)”, <http://lcg.web.cern.ch/>
- [19] Krishnamurthy S., Sanders W.H., Cukier M.: “A Dynamic Replica Selection Algorithm for Tolerating Timing Faults”, In *DSN* 2001.
- [20] Locasto M.E., Sidirolglou S., and Keromytis A.D., “Application Communities: Using Monoculture for Dependability”, In *HotDep* 2005.
- [21] “LLCbench: Low Level Architectural Characterization Benchmark Suite”, <http://icl.cs.utk.edu/projects/llcbench/>

- [22] “OSG”, <http://www.opensciencegrid.org>.
- [23] Raman R., Livny M., Solomon M.H., “Matchmaking: An extensible framework for distributed resource management”, In *Cluster Computing*, Vol 2, pp 129-138
- [24] “TeraGrid”, <http://www.teragrid.org/>
- [25] Tsouloupas G., Dikaiakos M., ”GridBench: A Tool for Benchmarking Grids”, In *IEEE Grid2003*.
- [26] Tsouloupas G. and Dikaiakos M.D., “Ranking and Performance Exploration of Grid Infrastructures: An Interactive Approach.”, In *IEEE/ACM Grid 2006*.
- [27] Vlachos M., Hadjieleftheriou M., Gunopulos D. , Keogh E., “Indexing multi-dimensional time-series with support for multiple distance measures” In *KDD 2003*.
- [28] “WISDOM”, <http://wisdom.eu-egee.fr/>
- [29] “Site Functional Tests (SFTs)”,
http://goc.grid.sinica.edu.tw/gocwiki/Site_Functional_Tests
- [30] Zeinalipour-Yazti D., Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos, N. Koudas, D. Srivastava, “The Threshold Join Algorithm for Distributed Top-k Queries”, In *DMSN 2005*.