AGrid: A Two-Tier Architecture for Grid Resource Management Middleware

Yoav Levy and Yitzhak Birk

Technion, Electrical Engineering Faculty
Haifa 32000, Israel
{yoavlevy@tx,birk@ee}.technion.ac.il

Abstract. In a typical Grid system middleware, the Resource Management Subsystem (RMS) is responsible for the provisioning of resources in Grid-attached computer clusters.
To date, intra-cluster short-term fluctuations in resource availability have not been effectively managed by existing RMS, mainly due to the inter-cluster communication latency preventing prompt dissemination and processing of resource (in) availability information. This leads to resource under-utilization that may downgrade overall Grid performance, especially regarding short, time-critical jobs.
We present AGrid[1], a two-tier software architecture for resource allocation and job scheduling middleware based on a network of interacting autonomous agents. While the upper tier performs global (inter-cluster) matchmaking based on global, semi-current information, lower-tier cluster-resident agents instantly react to local changes in resource availability by negotiating updated resource usage bids. We also offer a software framework that facilitates development of AGrid realizations, and demonstrate it by way of a simulative prototype.

Key words: Grid, RMS, Software Agents, Matchmaking

1 Introduction

Grid computing has matured in recent years and has transformed from a research domain into a viable compute solution for researchers and industry members alike. Grid systems are often compared to utility Grids [1], emphasizing its availability to everybody anytime and the fact that the user does not know or care exactly ehat resources he is actually using. Grid computing research enjoys a large body of work that ranges from fundamental manifests [2], [3], a large standard base of protocols and specification languages [4], and most importantly, a clear and consistent demand from industry for a viable alternative to multi-million dollar super-computers.

Grids may be considered as the next step in computing [5], following the eras of supercomputers and minicomputers, the advent of personal computing, and proliferation of cluster computing. Grids resemble compute clusters in terms of

[1] Agent-Grid

their ability to serve a heterogeneous group of users; their ability to gracefully degrade in performance when some of its components fail, and their good cost-to-performance ratio. However, Grids are unique as they may be composed of thousands and even millions of compute nodes (e.g. SETI-at-Home [6], be geographically dispersed, and feature a loose coupling between their components.

These traits have largely influenced the development of Grid middleware - the collection of monitoring and coordination subsystems provisioning Grid resources to user-submitted jobs. Unlike resource managers and job schedulers operating within compute clusters, Grid nodes enjoy a greater level of autonomy, and leave the middleware to perform coarse resource allocation and job management. One way for the middleware to offer a sustained high level of performance is to submit a job to a (very) large number of redundant Grid nodes, assuming some will fail unexpectedly.

1.1 The Resource Management Subsystem (RMS)

The (RMS) is a crucial component in every Grid middleware. The RMS is responsible for the following:

- **Resource and Policy Specification and Enforcement**: as a Grid node is composed of resources that originally belong to individual machines or clusters, these resources need to be re-categorized and specified in a platform-independent format to permit joint consideration of multiple such resources at the Grid management level.
- **Resource Monitoring**: resource availability (historic and projected / planned) is either recorded in repositories to be asynchronously queried by remote matchmakers, or reported to centralized matchmakers / schedulers. For example, a disk resource must be constantly monitored in order to ensure that its quota is not exceeded, that it does not experience intermittent or complete unavailability, etc.
- **Resource Allocation**: when jobs are matched with suitable resources, the RMS is responsible for receiving the desired assignment and allocating the resources to jobs. Allocations are to be monitored (for cases where jobs fail or need additional resources) and deviations should be mitigated.

As an example, Figure 1 below depicts the location of the Grid Resource Allocation and Management (GRAM) in the Globus architecture [3]. The GRAM component in each Grid node performs intra-cluster resource management for local jobs. A new job is received by the local gatekeeper and is handled till completion by a dedicated job manager that is responsible for communicating with the client that submitted the job. Both the client and job manager have coarse job management ability: the job manager's visibility of resource availability is reduced to the local Grid node, while the client suffers from Internet-class latencies that prevent immediate reaction to change in resource availabilities.

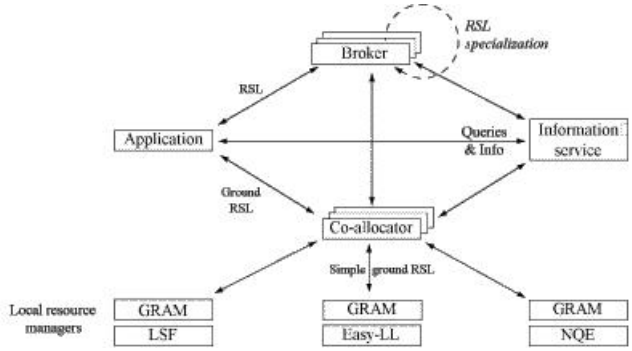

Fig. 1. Grid Resource Allocation and Management (GRAM) Model

1.2 Problem Domain and Categorization

In our research we have focused on Grid systems comprising of large sets of compute clusters interconnected by a WAN. This configuration was chosen since:

1. Computer clusters are expensive to procure and maintain, making their owners / administrators sensitive to the manner in which Grid jobs are submitted and executed. Owners seek to maximize resource utilization, and are able to publish resource usage policies and status.
2. Unlike a desktop machine, a compute cluster as a whole does not have idle cycles, and therefore cannot provision for idle-cycle stealers like Condor [7]. Grid jobs must be allocated resources and execute concurrently with locally-initiated jobs.
3. Globalization trends contribute to the proliferation of geographically-dispersed networks. It is not uncommon to see Grid systems, even intra-organizational, that span several continents, crossing numerous administrative domains and network subnets.

Also, we have turned our attention to prospective Grid users that would:

1. Submit shorter and more mission-critical jobs. For example, researchers performing experiments using nuclear accelerators work under tight schedules to crunch the vast amounts of data generated before their next run [8]. Another use may be in the field of interactive design, e.g. in the automotive industry.
2. Demand a higher level of stability and predictability. As Grid computing advances closer to the Utility Computing vision, more industry sectors view Grids a feasible data-processing alternative. Sectors like banking and aeronautics have much higher demands regarding the performance and durability of the computing infrastructure being used [9].

Moving from demand to supply, we identified two main deficiencies of contemporary middleware, both stemming from the centralistic character of existing architectures:

1. **Network Latencies**: The WAN links interconnecting the Grid nodes to the centralized matchmaker and scheduler feature latencies that are several degrees of magnitude higher than those of intra-cluster interconnects. Text-based network protocols used by Grid middleware (e.g. SOAP [4]) only expand this gap. Having a typical roundtrip delay on the inter-cluster links in the order of seconds defines a lower bound on the ability of the matchmaker / scheduler to respond to fluctuations in resource availabilities.
2. **Information Flooding**: The extensive standardization effort associated with Grid resource management [10], [11] has contributed to the abstraction of the heterogeneity in structure and behavior of the underlying resources. Alas, standard network protocols and resource specification languages are actually restrictive in their ability to disseminate fine-grained data through the distributed middleware.

The inability of a centralized matchmaker to effectively respond to fluctuations in resource availability in a timely manner downgrades the overall RMS performance. This adds to the scalability issue that arises when the centralized matchmaker is forced to enlarge its matchmaking compute cycle interval, as the number of jobs and participating clusters is growing. The combined effect may diminish the added value of Grid systems compared to localized compute clusters.

1.3 Research Objective

Our goal was to develop a Grid middleware architecture and implementation-supporting software framework, that together facilitate the specification and implementation of resource allocation policies and mechanisms that overcome the aforementioned shortcomings of prior art, as well as their evaluation when operating under various network and resource (in)availability conditions. Parties implementing a Grid middleware based on the AGrid architecture would enjoy a shorter prototyping cycle thanks to the software architecture, and would be able to integrate and evaluate various inter- and intra-cluster matchmaking algorithms.

2 The AGrid Architecture

2.1 A Two-Tier Model

We present AGrid, a resource management model to be deployed at two levels, or tiers (see Figure 2):

The upper tier is the Marketplace. It is implemented as a centralized process, typically running on a powerful and reliable machine or cluster with a high-speed Internet connection. It accepts job submissions and resource availability reports, performs periodic matchmaking, and publishes resource allocation plans. Having a global view enables it to balance load across clusters and match jobs to suitable resources.

At the lower tier, each compute cluster participating in AGrid maintains additional processes that perform local matchmaking. Each such process has two interfaces:

- An interface with the (local) cluster manager process. This is used for retrieving information on up-to-date resource availability, updated resource policies, and forecasted resource usage plans. Being local and closer (in network latency terms) to the jobs and resources enables the intra-cluster monitoring and matchmaking process to feature faster response and finer tuning with no scalability problem.
- Am interface with the (global) Marketplace. This is used for submitting resource availability digests, and to retrieve job submissions / updates.

Combining the two tiers enables quick, scalable fine tuning concurrently with slow, flexible global allocations.

The job perspective: An arriving job goes to the marketplace and is assigned resources based on global knowledge. Subsequently, it is directed to a specific cluster manager and is dynamically allocated resources in the cluster, dynamically competing with other jobs. Upon termination or a dramatic reduction in level of service, it returns to the marketplace in order to be reassigned. Figure 2 depicts a schematic diagram of AGrid.

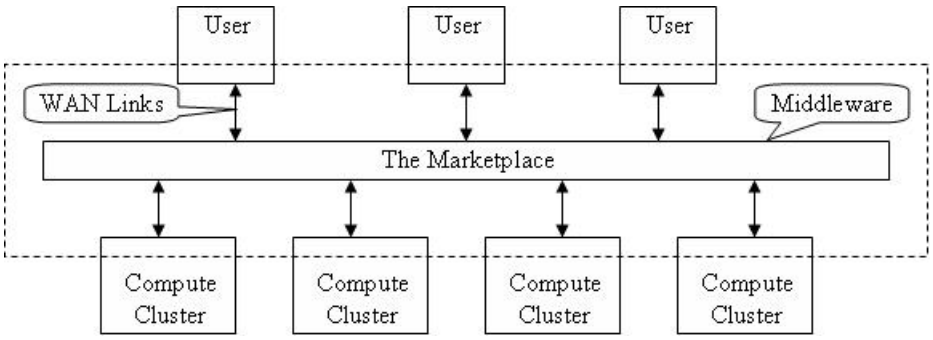

Fig. 2. Users, the Marketplace, and Clusters in AGrid

2.2 Persistent Agents

Application of software agents for distributed computing and Grid systems in particular has been studied and implemented [12] [13]. The autonomous and learning nature of software agents is suitable for large, loosely coupled and managed systems such as Grids. In AGrid, The distribution of processes and algorithms is implemented through the use of software agents. Users are represented by agents at the Marketplace. Each of the clusters hosts local user agents, referred to as proxies. These may communicate among themselves, with the local

cluster manager, and with the Marketplace using message-passing protocols. See Figure 3 for the layout of agents throughout the arena.

The benefits of having persistent software agents residing in compute clusters as well as in the Marketplace are:

- An agent may be instantiated unrelated to job submission events, and may continue to operate after a specific job has been completed or migrated. This prolonged lifetime may enable a specialized agent to gather more data into a cluster-local database of resource past performance available to other agents at that cluster, leading to better (tentative) allocation decisions.
- Agents of different types (user-representing and user proxies in our case) share a common infrastructure and protocols. This enables them to communicate and exchange information over various infrastructure settings and network conditions.
- The natural encapsulation of state and capabilities within each agent facilitates a more robust deployment as well as operation within heterogeneous collections of compute clusters.

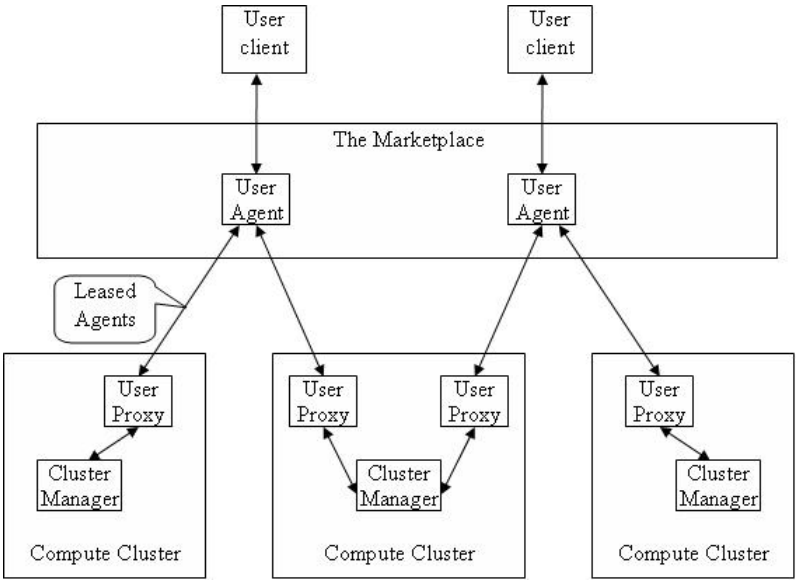

Fig. 3. Agent Overlay Network

Software agents are leased upon demand by interested parties. User Agents are leased by user clients if and when the user wishes to gain access to AGrid resources. This lease may hold as long as the user has more jobs to submit to the Grid, although it is feasible for an agent to remain active even if not leased. In

the same session-like manner, User Proxies in the lower tier are leased by User Agents.

This lease-based interaction mode enables agents to maintain relations that go beyond the scope of a single job, thus enjoying accumulated knowledge, while at the same time allowing reuse of software assets.

2.3 Grid Resource Allocation Using Computational Economics

Just like Utility Computing, the global, quasi-egalitarian nature of large Grid systems has drawn research that investigates application of usage models taken from the "old economy" [14]. Bids-based economics is regarded applicable for Grid systems where jobs need to be prioritized, but no centralized (human) authority exists to decide. Models and systems have been devised and implemented; a notable example is Nimrod/G, a Grid middleware based on the Globus framework [15].

Matchmaking and allocation in AGrid are performed in a negotiation-based, competitive manner. Based on resource availability and current allocation reports (published regularly by the Cluster Manager agent and User Proxies in the local clusters) agents construct bids per (possibly partially) available resource time slots that are candidates for allocation to pending jobs. These bids are sent to the Cluster Manager, where they are collected and evaluated, and are also sent to central Marketplace matchmaker where they affect future allocations (See Figure 4).

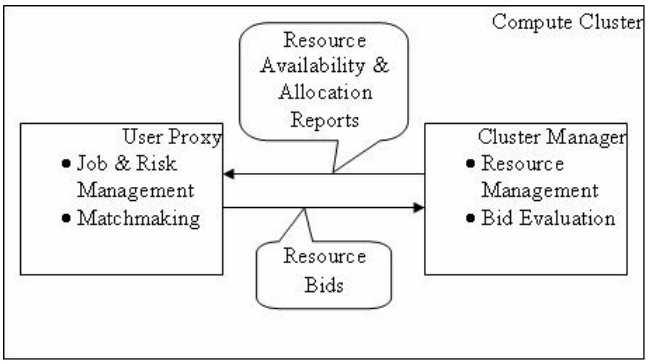

Fig. 4. Intra-Cluster Agent Interaction

The AGrid Marketplace process keeps track of every pending job, as well as of available resources in each compute cluster. The matchmaking algorithm attempts to find allocations of jobs to resources in a manner that will maximize overall utility. This allocation map translates into job dispatches (for newly submitted jobs) or job migrations (for already executing jobs for which better allocations were identified).

Having User Proxy agents in close vicinity of every Cluster Manager allows the above process to be conducted and repeated at a very high rate within each cluster, enabling the middleware to promptly react to risk / opportunity conditions within each cluster as they materialize, regardless of network latency and with no scalability problem.

2.4 Policy Design and Employment

AGrid policies are implemented as two-dimensional functions. This method was chosen as it is simple for end-users to understand and maintain. The writer of the function is required to break the function domain (e.g. resource cost) down into a collection of contiguous intervals. Per each interval, the writer then defines a polynomial function by specifying its coefficients. Below is an example of a utility-time relationship. According to the specified function, utility decreases as time progresses, diminishing below zero around time of 350. The formula corresponding to this function would be:

$$1.2 * X^0 + (-0.0001) * X^{2.6}$$

2.5 Example

Consider two prospective AGrid users: a researcher in a rush to meet a deadline and a freshman student looking for a faster alternative to his home computer. The researcher will employ policy that reflects a willingness to pay higher fees for the prompt completion of the batch, while the freshmen has no budget to spend and would be grateful to have his work done by next morning. Both specify their respective preferences in policy profile documents, to be attached to / referenced by jobs they submit from now on to the Grid.

The marketplace matchmaker performs global resource matchmaking, assigning each job to a cluster(s). Final resource allocation (within each cluster) will be made by each cluster-resident agent based on the user policies attached to jobs arriving at its cluster, and most current resource state.

Suppose next that resources allocated to the batch job are experiencing intermittent malfunctions. The local agent will respond by attempting to reallocate (potentially more expensive) resources in order to ensure meeting the contracted completion deadline. At the same time, the unstable resources now de-allocated may be immediately allocated to service (pro-bono) the freshman's job.

After a short while, the marketplace matchmaker will be notified by the proxy agent of the state of the lagging batch job. In return, the marketplace matchmaker will use this information combined with information it receives regarding resource availability in other clusters to decide on a migration of the job to another cluster.

As soon as such a decision is reached and the batch job has been checkpointed, the local agent is able to allocate the freed resources to the freshman's job without any additional delay.

3 Enabling Software Framework

To support the introduction of AGrid to the Grid research community, and to facilitate future implementations of AGrid as part of an industry-class middleware, we accompany the model described here with a tested software framework. It consists of:

- A code generation capability that enables fast prototyping starting from an XML entity definition. The entity library includes the collection of agents, clusters, resources, users and jobs. Once generated, each entity definition contains its configurable data (values modifiable via external configuration files), properties (state variables), and capabilities (entry points to code performing tasks).
- A managing layer responsible for simulating the network and users, and generate / dispatch time-driven events.
- A messaging infrastructure enabling simulative as well as real-life distributed inter-agent communication.
- An algorithm container enabling the execution of pre-integrated resource allocation and job scheduling algorithms.
- A configuration schema that defines the collection of clusters and Grid users, the jobs to be submitted, and behavior of resources in terms of availability and reliability.
- An evaluation and benchmarking suite enabling the definition and execution of tests, including simulative network configuration and job generators.

We chose to implement the framework for AGrid using the Java programming language, which is supported on a variety of platforms and is ready for code generation.

Once the framework user specifies simulation configuration data and executes the framework's main class, the managing layer initializes the network, cluster, user and marketplace objects. From there, each object is invoked to perform its implemented capabilities based on its state (e.g. pending jobs, resource preset behavior). Messages are generated and dispatched via a centralized queue. performance data is logged throughout the simulation for debrief purposes. The pre-integrated algorithms are periodically executed and allowed access to the local state of the agent object that hosts them. The simulation terminates either when events are no longer fired, or when a time limit is exceeded.

A prototype of the framework has been produced and was used to conduct a preliminary performance evaluation. The data gathered from several benchmarks summarized in the appendix suggest that AGrid performance is better when short jobs are submitted, and when the level of resource availability fluctuation is high.

Users of this framework will be able to shorten the time it would take them to implement a working two-tier multi-agent prototype, and allow them to concentrate on the development and integration of various matchmaking and scheduling algorithms.

4 Conclusion and Future Work

This work has focused on the behavior of Grid resource allocation and matchmaking middleware operating under harsh network latency and resource unavailability conditions in cluster-based Grid nodes. Performance of existing middleware degrades when above conditions deteriorate, leading to waste of expensive resources.

In this work we have proposed a resource matchmaking and allocation middleware model and accompanying software architecture that facilitate fast development of marketplace and proxy agents, user and resource-owner policies, as well as integration of matchmaking algorithms. It was argued that the agent-based, two-tier model overcomes the fundamental problem of contemporary middleware: the inability to harmoniously accommodate a global inter-cluster matchmaking of Grid resources to user jobs and a fast-adjusting intra-cluster resource allocator. In order to ease and encourage the development of software implementations that conform to the AGrid model, we developed a Java-based software architecture and building blocks offering prospective implementers a kick-start and allowing them to focus on applicative issues such as matchmaking algorithms and resource usage policy design.

A detailed quantitative assessment of the operation of AGrid and its advantage over contemporary middleware is beyond the scope of this work. It requires implementation of the model over an existing RMS system components that would facilitate access to the underlying resources and Grid communication protocols, and integration of a market-class matchmaking algorithm.

Further research in the area of Grid resource management can be facilitated by the development of an AGrid-conforming middleware. Suggested research venues include user and cluster owner policies, application of bid-based matchmaking algorithms, and collection and dissemination of resource usage data in a distributed manner.

References

1. Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
2. Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.
3. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Published online at `http://www.globus.org/research/papers/ogsa.pdf`, January 2002.
4. World Wide Web Consortium. *Simple Object Access Protocol (SOAP) 1.2*, 2003.
5. Ian Foster and Carl Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
6. David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti-at-home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

7. James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 55, Washington, DC, USA, 2001. IEEE Computer Society.
8. Jeremy Coles. The evolving grid deployment and operations model within egee, lcg and gridpp. In *e-Science*, pages 90–97, 2005.
9. Charles Brett and Robert Cohen. Applying the grid to commerce. Published online at `ftp://ftp.cordis.lu/pub/ist/docs/grid/grid_mws_0212.pdf`, 2002.
10. ZhenChun Huang, Lei Gu, Bin Du, and Chuan He. Grid resource specification language based on xml and its usage in resource registry meta-service. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 467–470, Washington, DC, USA, 2004. IEEE Computer Society.
11. Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, Candidate Recommendation CR-wsdl20-20060327*, March 2006.
12. Junwei Cao, Daniel Spooner, James D. Turner, Stephen Jarvis, Darren J. Kerbyson, Subhash Saini, and Graham Nudd. Agent-based resource management for grid computing. *ccgrid*, 0:350, 2002.
13. Junwei Cao, Darren J. Kerbyson, and Graham R. Nudd. Use of agent-based service discovery for resource management in metacomputing environment. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 882–886, London, UK, 2001. Springer-Verlag.
14. R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. Hey, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.
15. Rajkumar Buyya, David Abramson, and Jonathan Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid, 2000.
16. Rob F. Van der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. In *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01)*, page 315, Washington, DC, USA, 2001. IEEE Computer Society.

Appendix: Preliminary Performance Evaluation

As a proof-of-concept, the above architecture was prototyped in a working simulative framework. This prototype was exercised as a test-bed for initial evaluation of exemplary policies and matchmaking algorithms.

The simulative prototype was tested using the NGB (NAS Grid Benchmark) specification [16]. A three-dimensional test matrix defines three aspects of test variations: the inter- and intra-job complexity as defined in the NGB specification; the level of change in inter-cluster network latency and resource availability; and disabling intra-cluster competitive matchmaking.

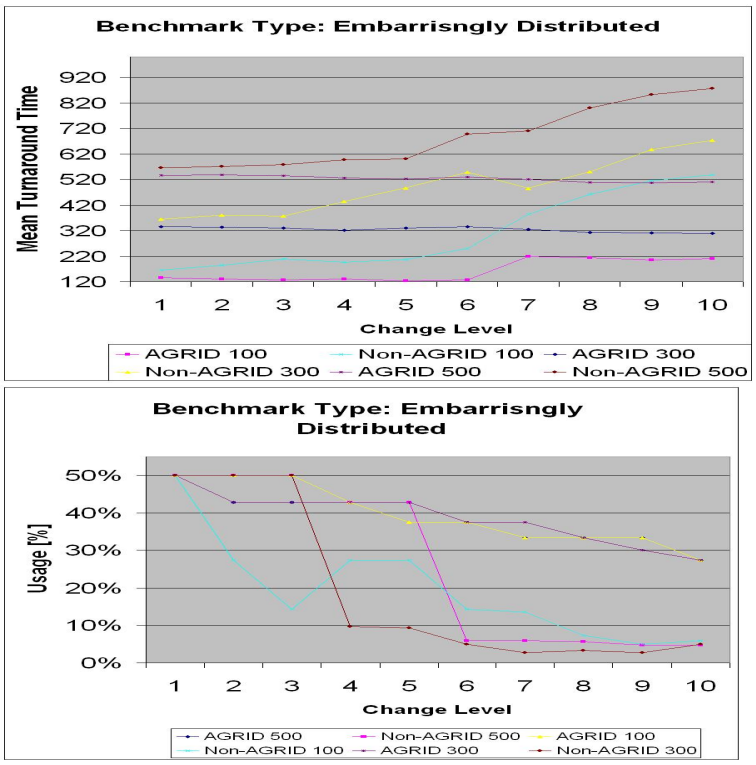

Fig. 5. Embarrassingly Distributed Turnaround Time and Resource Usage

The bench-marking tests have provided the following initial findings:

– **Effect of Change Level on job Turnaround Time and Resource Usage Levels**: While increased change level increases the turnaround time, Figure 5 shows that AGrid performance gracefully degrades under the three change levels (100, 300 and 500) from 50

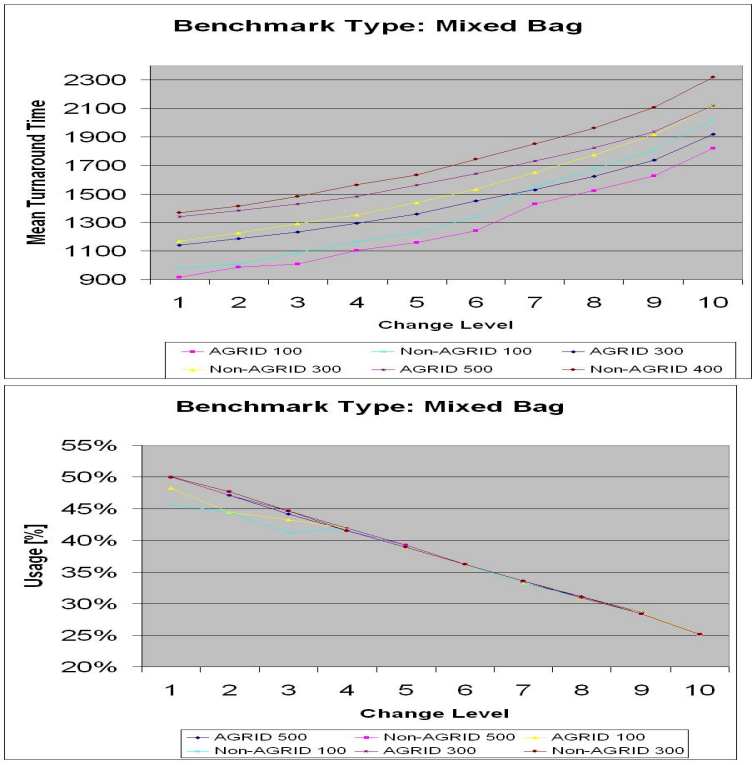

Fig. 6. Mixed Bag Turnaround Time and Resource Usage

– **Sensitivity of Turnaround Time to Job Size and Complexity**: The four job types defined as part of the NGB benchmarks represent progressively larger and more complex jobs. Observing the effect of turnaround time per each job across various change levels, the difference between AGrid and the simulative reference system (Non-AGrid) diminishes. While the single-segment ED benchmark exhibits a totally different behavior between the two simulated systems, the MB benchmark exhibits a very close behavior with minimal improvement in turnaround time. This result is to be expected: large and very large jobs do not benefit considerably from a cluster-level distributed matchmaking mechanism, as the centralized matchmaker is able to kick in at a relatively reasonable rate to reschedule lagging jobs (Figure 5 vs. Figure 6).