Preliminary version; to appear in 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Sept 19-21,2007, Austin, TX

Meeting Virtual Organization Performance Goals through Adaptive Grid Reconfiguration

Zach Hill*, Jonathan Rowanhill*, Anh Nguyen-Tuong*, Jim Basney**, Glenn Wasson*, John Knight*, and Marty Humphrey*
*Department of Computer Science, University of Virginia
**National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign

***Abstract*— In order for Grids to become relied upon for critical infrastructure and reliable scientific computing, Grid-wide management must be automated so that it is possible in quickly and comprehensively respond to or anticipate specific environmental changes and requirements. That is, due to the disjoint administration of Grids which results in high communication requirements as well as large numbers of skilled administrators, manual reconfiguration of large portions of a Grid in such situations is not a viable solution. We have developed an architecture that allows changes to the grid configuration to be automated in response to operator input or sensors placed throughout the Grid. Additionally, our architecture allows the resource owner to specify who is allowed to alter the configuration of his resource and what types of reconfigurations are allowed. This permits the automation of reconfiguration while retaining owner control of the individual resources. Our experiments show that this architecture can be used to reconfigure a representative GT4-based development grid in order to enforce maximum latency limits on a Virtual Organization's jobs by dynamically controlling job priorities Grid-wide.**

I. Introduction

Managing the day-to-day operation of a grid is a very challenging task. Current grids can require a significant effort to monitor the availability and performance of the grid software and to make changes. Since grids often cross administrative domains, the time required to affect changes to a significant portion of a grid can take days, weeks, and sometimes months, usually through email or telephone exchanges, or direct exchanges between human operators.

Software updates, reconfiguration of the middleware and/or application layers, security patches, and modification to scheduling policies are just some examples of grid management. Ordinarily, these tasks are left to the individual resource owners who have to make changes in the time and manner that best suits their organization. At any point in time, therefore, the grid is in a potentially inconsistent state of modification. As a result, the user might not be able to rely upon the grid delivering its claimed service levels.

More precisely, the problem with the existing largely human-based management of grids is in part grid integrity and correctness and in part the relatively high latency of effecting change when viewed holistically. For example, a security vulnerability in a grid must be corrected in *all* resources and sites affected, and this must be accomplished *quickly*. Some grids lack the formal process by which to plan and accomplish such a grid-wide response. In fact, the site-autonomous nature of grids arguably facilitates this lack of systemic response.

Some grids (e.g., TeraGrid [1] and the Open Science Grid [2]) have a more structured process in place, organized hierarchically. For example, the Open Science Grid Governing Board can set policy overall, which then passes to the governing boards of the individual VOs, then to the site-level managers. However, at each point in the hierarchy, it is not clear when or if such information is either acted on or passed along.

We claim that much of this human-based grid management can and should be replaced or at least augmented by a software-based management structure. In general, software would be written and deployed to monitor the grid and effect changes as a result of observed behaviours. A key challenge would be the construction of the logic by which to map observed behaviours to actions designed to modify the collective actions of the grid. A "semi-automated" approach might be taken in which key decisions should require human confirmation. If successful, the benefits include more consistent application of policy, more precise and complete records of decision making (e.g., for *post-facto* auditing), and overall faster response times.

This paper presents a comprehensive and automated software-based grid management system. Our approach relies upon a hierarchical control loop architecture composed of sensors, actuators, and rule-based control logic. These control loops are organized into a hierarchy such that the global, grid-wide control resides at the top with VO-level control below and potentially site-level control below those. Thus separate rules can be enforced at each level with different policies in place for each VO and site. Specifically, the contributions of this paper are:

- A system architecture to reconfigure a grid dynamically without human intervention in response to environmental changes.
- An implementation of the architecture in Java and integrated with Globus Toolkit v4 (GT4).
- A collection of policies to enforce a quality of service constraint across the grid for a virtual organization.

We evaluate and show the utility of the system through a case study in which a small GT4-based development grid is dynamically reconfigured on a per-site basis through the

This material is based upon work supported by the National Science Foundation under Grant No. SCI-0426972.

application of site-specific and then VO-specific control logic, so that the VO is then able to meet its performance goals.

The remainder of the paper is organized as follows: section II describes related work; section III presents our management architecture and its integration into the grid; section IV is an experimental evaluation of our architecture, implementation, and example policies; section V is a discussion of pertinent issues; and section VI gives conclusions from our work.

II. Related Work

There exist many hierarchic management architectures in the literature; some commercial and others experimental. The size of the managed systems and the scope of management vary with architecture. Applying the model of Martin-Flatin et al [3], we can describe management architectures by the nature of their delegation capabilities.

Early management architectures, such as traditional SNMP-based systems [4], apply centralized control. SNMP-based systems have evolved to apply management hierarchy. Examples of hierarchic management systems include the SAMPA [5] architecture, the Tivoli Management Framework [6], and the Open Systems Interconnection (OSI) standard [3]. In these, the power of late-bound, run-time delegation service is increased, allowing managers to delegate authority through commands over otherwise static infrastructure in static hierarchic configurations.

The overall goals of our project overlap with the ambitious vision of autonomic computing [7][8]. By drawing parallels to the human autonomic nervous system, the intent of autonomic computing is to create software systems that can manage themselves with minimal human intervention. The research described in this paper has similar goals to many existing autonomic computing projects; goals that include self-awareness, self-healing, anticipatory response, and so on. The Willow approach that we discuss here is the first attempt at controlling a grid (e.g., GT4) via an explicit management architecture that incorporates multiple, hierarchic sets of semi-independent control logic.

Sensing state is common to many grid projects, mainly to provide human operators with important information. Hawkeye [9], SCALEA-G [10], RGMA [11], GridMon [12], and INCA [13] are systems that gather data about the state of components of the grid, including the network infrastructure. MonALISA [14] provides complex sensing capabilities and provides an interface with which a grid administrator can interact and reconfigure resources.

The actuation component of our architecture is most similar to grid software systems such as the Virtual Data Toolkit (VDT) [15]. VDT is intended as a software release and update tool much like *yum* or *rpm*. None of these systems are designed to provide automated responses. In the work described in this paper, we did not utilize an existing monitoring system such as VDT, mainly because of the complexity of integration. A simpler, less comprehensive mechanism was sufficient for the research described, without precluding the use of one of these tools in the future.

The experiments and scenario conducted in the paper involve shifting resources and in general reconfiguring the grid in order to achieve performance-related goals on a per-VO basis. The closest project in the grid community to address similar performance-related goals is the TeraGrid SPRUCE [16] project, in which special priorities are pre-defined and pre-authorized for "urgent computing". The goals and application of the Willow approach to managing grids described in this paper is broader than SPRUCE; for example, our approach features continual sensing of the state of the grid, with multiple (sometimes conflicting) independent control logic for re-posturing the grid dynamically (e.g., by identifying and applying software patches as necessary to limit known vulnerabilities).

III. Management Architecture

A. Requirements

One way to approach the problem of creating a software system for managing a grid is to consider each existing software component on a per-component basis, modify the component to facilitate monitoring and dynamic behaviour modification, and then attempt to re-compose the overall functionality using the modified parts. This is particularly unrealistic for grids, given the relative maturity of the software components measured in part by active deployments. In essence, it is impossible to redesign the existing grid components for these capabilities, and it is similarly impractical to suggest holistic replacements for existing parts.

Instead, our general model is that there is a separate hierarchical control structure which is integrated with an existing grid system's sensing and actuation mechanisms. The control logic is implemented solely within this control structure, not within the existing grid software. This decoupling allows both the control logic and implementation to be developed and to mature semi-independently of evolving grid functionality.

Such a software system for semi-automated management of large-scale grids has both "situation-independent" and "situation-dependent" requirements. A grid management system can be described in terms of its specific intended purposes, e.g., ensuring a particular VO meets its performance goals, or ensuring that all available software patches have been applied to grid nodes.

It is useful to break the requirements of a grid management system into those that are generic to the entire grid management system and those that are specific to a single purpose. The purpose-independent requirements include: (a) the system must acknowledge, support, and balance holistic grid behaviour goals against the local self-interest of nodes; (b) the system must scale; (c) the system must have reasonable overhead; (d) the management system itself must not be an unreasonable management burden; and (e) the management system must support the composition of purpose-specific management techniques.

The purpose-specific requirements of a grid management system include: (a) relatively easy integration of local,

purpose-specific performance measurement techniques into the grid management system, e.g., providing the user the ability to specify: how and when to initiate the performance measurement technique, how the management system should interpret the results of individual performance measurements, and how to aggregate individual sensor measurements; and (b) similarly easy mechanisms for effecting management changes, e.g., providing the user the ability to specify: software mechanisms to effect change; how and when to initiate these software mechanisms; and how to interpret their results.

B. Overview of our Approach: Willow, ANDREA and Spartan

To meet these requirements, we leverage and extend the Willow architecture [17] to meet the challenges of a management architecture for large-scale grids. Willow is a comprehensive architecture that provides survivability for large-scale distributed systems. It consists of a nested hierarchy of control loops that sense and analyse the state of the system, and a powerful network management mechanism to effect the required changes on the underlying system.

As shown in Figure 1, the two major pieces of Willow are *Spartan*, which provides the state sensing and analysis capabilities, and *ANDREA*, which provides the network management capabilities. In the remainder of this section, we describe Spartan and ANDREA in more detail [18][19][20].

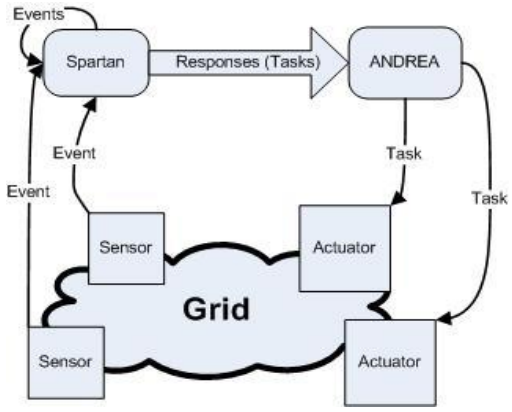

Figure 1. Willow Control Structure

Spartan, the sensing and analysis functionality of the Willow architecture, is a hierarchy of finite state machines that process sensor events and determine actions based on state definitions and transitions. A Spartan node is one or more finite state machines that receives sensor input events either from other Spartan nodes or directly from sensors on the target resource (the grid in our case).

It is the hierarchy of finite state machines that allows Willow to provide a wide range of different control capabilities. The hierarchy is established for any specific system in such a way that it matches the overall management goals of the system. Thus, for example, the top-level finite-state machine in the hierarchy will be concerned with grid-wide issues, the next level with issues faced by regions of the grid, and the lowest level with issues faced by local elements of the grid, perhaps a single site.

Where conflicts arise in determining management actions between the different levels, the Spartan system effects a prioritization based on a separately provided specification. Thus, the management system can be constructed easily to ensure that grid-wide management changes take precedence over local changes no matter how they conflict, but consistent local changes will still be applied.

Specification of the control mechanism for a system is written in the Time-based Event Detection Language (TEDL) [20], and the resulting specification compiled into an appropriate implementation. Using this mechanism allows management policies to be changed quickly and the changes implemented quickly since the generation of the implementation is automatic.

The basic component of TEDL is an event. An event is a time-stamped set of (*name, value*) pairs. A set in TEDL is defined as a collection of events whose time stamps fall within some defined range relative to the current time on the machine executing the finite-state machine. Predicates are then defined relative to attributes of sets, and predicates can be composed to form compound predicates. The change in value of a predicate during operation is the source of action and higher-level event generation.

A finite-state machine is defined by a set of states, each of which has actions associated with it as well as transitions between the states that are defined relative to predicates defined over sets of input events. As the machine transitions between states based on input events, actions are triggered corresponding to each state visited. Actions may be either to output an event to another finite state machine or to create a task and run that task on the system in order to alter the state of the system (a grid in our case).

Complex sensing and scalability are achieved in Spartan by hierarchies of Spartan nodes. At each level the sensor data is analysed and decisions are made relative to the local environment of each node. Events are transmitted to nodes higher in the hierarchy thus giving higher level nodes a more abstract view of the system and increasing the scope of the knowledge of nodes as they approach the root node.

The root node in a Spartan system sees the world in a very abstract way and can make determinations on courses of action by taking into account an entire large-scale distributed system without having to know who, where, or how many nodes there are in the system. Since each level of the hierarchy receives input only from the nodes directly below it, it is possible to make intelligent decisions about large systems without each level having detailed knowledge of the rest of the system.

As the finite-state machine of a Spartan node transitions through states, it may output *tasks* to be performed in order to alter the configuration of the system. This is the only point at which Spartan is bound to ANDREA, the task generation and task passing interface. Once a Spartan node determines that a task should be executed, it creates it and passes it off to ANDREA to see that it is executed on the nodes of the monitored system. Data about the effect of the task is

gathered by Spartan from sensor events it receives after the task is executed.

ANDREA is a management system for very large-scale distributed systems. It is built on the premise that there will be more nodes in the system than can be tracked by any one entity and therefore it is based on a hierarchic structure. The hierarchy is imposed over a highly modified publish-subscribe communication network permitting nodes to determine what tasks they will accept and from whom they will accept them. Tasks are passed through the network of control nodes based on subscription attributes stating which sources and types of tasks a node is willing to accept. It is this mechanism that permits resource owners to determine what actions are taken on their resource. This is also the mechanism that permits ANDREA to control very large systems. In effect, addressing nodes within the system is based on attributes that the nodes have and not necessarily on specific network addresses or some other form of identification.

More specifically, a task in ANDREA is a *workflow*. ANDREA supports most typical workflow constructs such as delegation, sub-tasks, and conditional tasks (if-then-else, etc). Each task has associated with it a selector that implements the addressing mechanism discussed above. This selector field can be very specific, e.g., the name of a single node, or can be a general property of a node, e.g. any machine running Fedora Core 6. Thus, both sender and receiver nodes in ANDREA have the ability to filter tasks according to their specific needs.

Tasks in ANDREA are considered to be constraints on the configuration of a system. Enacting a task on the grid therefore equates to constraining the configuration of the grid to conform to some requirement. ANDREA holds the constraint until it is told to do otherwise. With regard to Spartan this means that when a Spartan finite-state machine enters a state, it enacts a constraint on the grid configuration and that constraint is not lifted until the machine enters another state that explicitly tells ANDREA to either further constrain the system or to relax the previous constraint.

IV. Experimental Results

In order to assess the feasibility, applicability, and efficacy of the Willow approach to grid management, we conducted a case study in which we applied the reconfiguration mechanism to a prototypical GT4-based development grid we deployed and maintain for experimental purposes. The scenario we used is small in scale but sufficiently organizationally complex to represent realistic environments such as the Open Science Grid (OSG).

In OSG-like environments there are multiple VOs and each site administrator may choose to support one or more VOs. This results in environments with competing and sometimes conflicting priorities and requirements. For example, if two VOs need to use the grid intensely at the same time, then a decision must be made as to who should get higher priority or if they should just compete for resources equally. It is in resolving conflicting demands such as this that the Willow architecture performs well because of its ability to express many situations and their corresponding requirements.

A. Scenario

In our scenario there is a grid that consists of three sites named UVA, NCSA, and SDSC. These names have no real meaning in our scenario other than to identify each site. We are not modelling any observed behaviour or predicted behaviour of these sites in the TeraGrid, for example.

Additionally, there are two Virtual Organizations present in our GT4-based development grid, modelling the environment of the Open Science Grid in which multiple VOs exist and resources support all or some subset of the VOs. We refer to one VO as *PHYS* ("Physics") and the other VO as *BIO* ("Biology"). Again, we use these names only for identification with no specific attached meaning.

In this scenario, members of both VOs are submitting jobs to the grid. The property to be enforced is that PHYS's jobs are temporarily more important than the jobs of any other VO. They have a performance goal that they must finish within a reasonable amount of time and should have priority on the resources. However, they are only more important for some relatively short time period, such as a day or a week. During this time, the PHYS VO will be submitting jobs to the compute nodes of all three sites at a uniform rate and the jobs must complete within a bounded time period.

During the period when the PHYS VO is submitting jobs the BIO VO is also submitting jobs, but not necessarily at a uniform rate. Specifically there will be a "normal" submission rate throughout the period of the PHYS submissions and there will also be periods of increased workload imposed by the BIO VO, but these increased workload periods will not be continuous. In our test, this increase in workload is represented by a period of increased submission rate at each of the sites. These periods do not overlap and thus exhibit a "roaming" characteristic by which the increased rate moves between the sites affecting each individually.

One potential approach to this problem is to view it as a simple scheduling problem, and to try to solve it by setting the priorities of the schedulers at each compute node so that the PHYS VO has the highest priority and will always get the resources first. However, although our development grid has only 3 sites with 5 compute nodes each (and thus only 15 compute nodes in the system), a production grid would have many more nodes making priority assignment a significant effort. Furthermore, each node might be administered by a different person, and so to enforce a global scheduling priority would require changes at each node individually. Thus, modifying individual schedulers is a particularly undesirable solution when the ordering of priorities changes dynamically and unpredictably. In our test we only determine relative importance between the two VOs once, but, as stated earlier, this is only a temporary ordering. It is entirely possible that two days later the BIO VO will need to have top priority.

A different approach to this problem is to mandate a grid-wide scheduler to which all jobs are submitted and where all

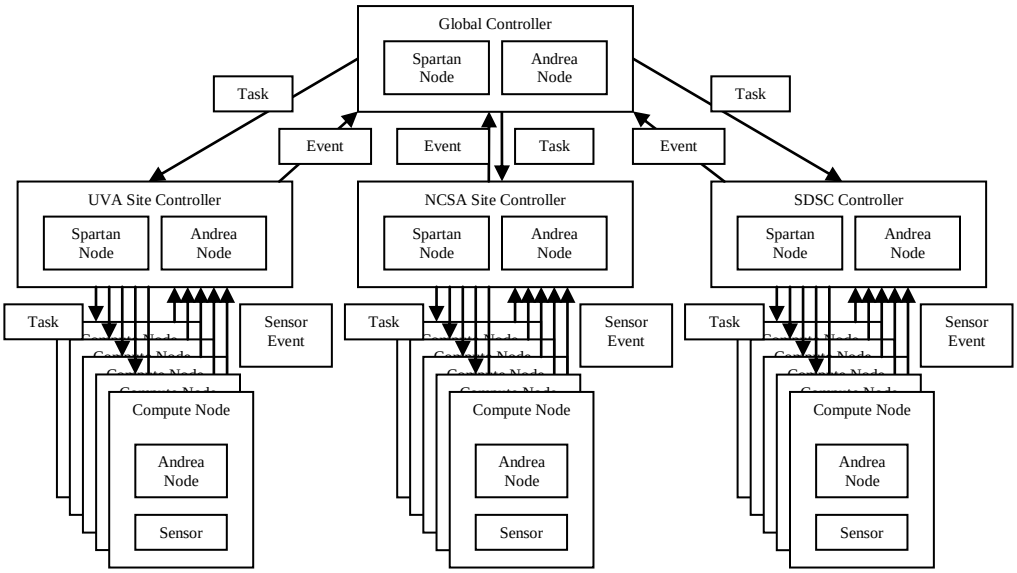

Figure 2. Willow Configuration on GT4-based Development Grid

priorities are enforced. The disadvantage of such an approach is that it results in a total scheduling order for all nodes in the grid and thus imposes unnecessary constraints on the resource owners in terms of defining policy. In addition, while there are many "user portals" deployed in grids today to present a customized look-and-feel for particular VOs, we know of no existing general-purpose and/or multi-use grids that attempt to restrict *all* users in this way in order to set a global scheduling policy. Potential grid users would almost certainly object to this one-interface-fits-all mandate. As well as being inflexible, it requires that a single node in the grid handle a large portion of the work, and this node becomes both a single point of failure and a potential performance bottleneck.

By leveraging and extending Willow, we approach this situation as a dynamic control problem where the sensors monitor the job turnaround time of the PHYS VO, and the actuators reconfigure the scheduling policy or process priorities at the compute nodes as necessary. Because Willow provides a hierarchy of control loops, it is possible to have multiple policies being enforced simultaneously at the local, site, and grid levels.

We note that the policies employed in a production system need to encompass more than performance goals of the type we investigated. For example, Willow would most likely be used to provide a recovery mechanism from serious, widespread hardware failures as well as managing performance.

Willow provides a degree of flexibility and adaptability not present in the other approaches. Even more importantly, it allows the actual reconfiguration to be done automatically and thus reduces the workload of individual site and resource administrators. The strengths and weaknesses of Willow and the other approaches are discussed further in Section V.

B. Experimental Setup

To implement the scenario described above, Willow was deployed on our GT4-based development grid of 15 compute nodes in a two-tiered control loop organization with a single global control loop and three site-level control loops, one for each site (Figure 2). The global- and site-level nodes in Willow each consist of an ANDREA node and a Spartan node implemented as Java objects and coupled programmatically to allow Spartan to tell ANDREA how to actuate the GT4-based grid components.

The concept of a site is implemented by assigning a *SiteID* attribute to the ANDREA nodes such that a set of five compute nodes and one site-level node share a common *SiteID*. Each compute node consists of an ANDREA node which includes that compute node's system actuator, a sensor and a GT4 installation. From Willow's perspective, only the sensor and ANDREA node are part of the compute node, the GT4 installation is not. Each compute node in the system (global, site, and compute) runs on its own physical machine, and the machines are connected via a GigE local area network.

It is important to stress that the only portion of the experiment that was truly simulated was the "job" that the grid was required to run (from both VOs), consisting of sorting a random array of integers. The job's only outputs are a pair of log messages, one at initiation of the sort and one at completion that includes the wall clock running time of the sort. The sort is essentially completely CPU bound and thus responds well to actuations that alter the CPU scheduling priorities. While we believe that our approach is suitable for actual VO jobs, we created such simple and parameterized jobs to simplify our testing and clarify the interpretation of the results.

Execution is initiated via GT4 job submissions and thus all jobs executed on a compute node output to a common log file. The sensors monitor the log file for the compute node upon which they are running, and they simply average the

running time of the last three completed jobs submitted by the PHYS VO user. Each sensor scans the log of its compute node every 30 seconds and sends the information as an event to the Spartan node for its site, thereby introducing the base-level events into the control loop.

C. Willow Configuration Details

As noted above, specification of the control actions that are required for each control loop in the Willow system are defined using the Time-based Event Detection Language (TEDL). The ANDREA WorkFlow Language (AWFL) is used to specify the details of the control actions that the system takes in the form of tasks described by workflows.

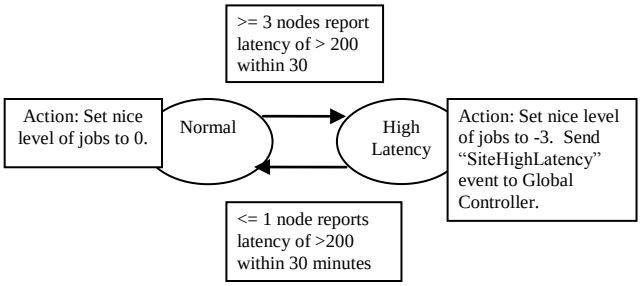

Figure 3. Site-level Specification

For the scenario described above, the specifications were as follows. The site-level TEDL specifications were identical for all of the site nodes in the system (Figure 3). The specification states that if, within a 30 minute window, three or more compute nodes in the site send events indicating that job latency for the PHYS VO is above 200 seconds, then the Spartan node should respond by: (a) initiating a task on the compute nodes to raise the priority of the PHYS VO user's jobs; and (b) send a *SiteHighLatency* event to the global controller indicating that the site is experiencing high job latency. The priority increase is implemented as a *renice* of the user's jobs to nice level -3.

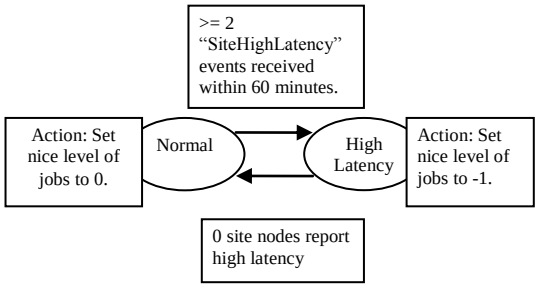

Figure 4. Global-level Specification

The TEDL specification for the global control loop specifies that if two or more latency events are received within a 60 minute window each from different site-level controllers, then the global controller should initiate a task to increase the priority of the PHYS VO user's jobs at all sites in the grid (Figure 4). The priority increase specified by the global task is to set the user's jobs to a nice level of -1. Thus, the global response is less severe than that of the site-level controller.

D. Simulated Workload Characteristics

The workload used for the tests consisted of the two VO users submitting jobs at a constant rate to each compute node at each of the three sites. Each user submits two jobs per minute per compute node for the 50-minute duration of the test for a total of 100 jobs per compute node.

At predefined offsets from the first job submission, a disturbance is created at each of the sites by the BIO VO. The disturbance is a brief increase in the rate of job submission and changes in the nature of the jobs themselves. The jobs submitted during the disturbances are submitted in pairs with one pair submitted per minute for three minutes. The jobs have running times that are more than twice the running times of the normal jobs being submitted.

The first disturbance is started at the UVA site compute nodes 10 minutes after the first normal load job is submitted. The second disturbance is started at the NCSA site compute nodes 10 minutes after the disturbance at the UVA site is started, and the third disturbance is started at the SDSC site compute nodes 10 minutes after the start of the NCSA site disturbance.

The GT4 GRAM service is configured to use the forking job manager, so jobs are forked into existence as they are submitted as opposed to being placed in a queuing system such as PBS or SGE. This is an important detail since it determines how jobs compete for resources and how newly submitted jobs can influence the execution time of jobs already running on the system. This is not the case with a queuing-system backend. Using a forking GRAM service is the reason why the disturbance jobs are few but long running; because they run for a long time they affect the latency of many jobs not just the jobs submitted after them. In addition, the long runtime of the disturbance jobs causes the normal load jobs to run longer and thereby begin to accumulate on the system.

E. Results

To evaluate how well our implementation of the Willow architecture is able to handle the scenario described above, we compared the results of running the workload with and without Willow operating. The metric of interest was the turnaround times of the jobs in the workload.

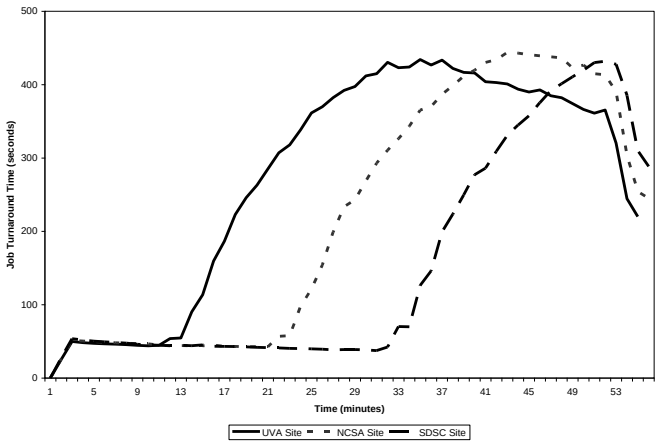

Figure 5. PHYS job turnaround times (Willow not running)

Figure 5 shows the turnaround time for the 100 job workload of the PHYS VO user with the grid operating normally (unmodified) as measured by the system's sensors. The disturbances are clearly visible on the graph as sharp increases in turnaround time. The staggering of disturbances is visible between sites. The disturbances start 10, 20 and 30 minutes after the start of the workload, but they do not manifest themselves until several minutes later since the jobs must complete before the sensors detect any changes in turnaround time.

Recall that the disturbances are the result of injecting relatively few high-demand jobs over a relatively short time. Despite this, the effects of the disturbances can be seen clearly for extended times. The turnaround times of the normal jobs remain high well after the disturbances are introduced.

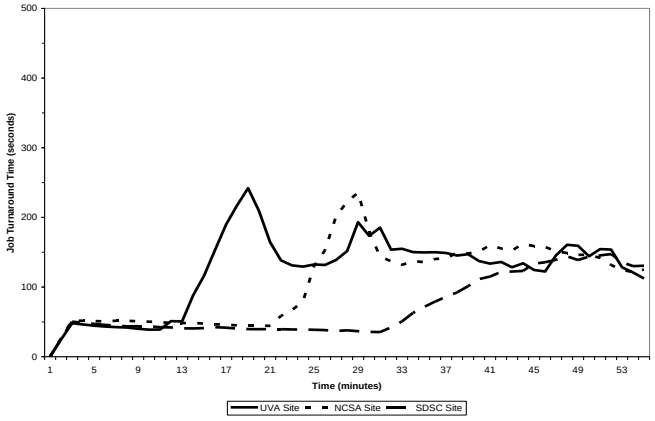

Figure 6. PHYS job turnaround times (with Willow)

Figure 6 shows the PHYS job turnaround times with the complete Willow system operating. Willow is able to reduce the turnaround times significantly following disturbance from those experienced in the unmodified system (Figure 5).

The results in Figure 6 show that the third disturbance, at the SDSC site, does not bring the turnaround time over the threshold of 200 seconds at all. This effect occurs because of the action taken by the global controller after the second disturbance. It has changed the priorities at the SDSC site proactively even though no disturbance had taken place at that site yet. This control policy is present to deal with possible multi-site problems. If two of the three sites report that turnaround times have exceeded a prescribed threshold, then the problem (whatever it is) is not local and cannot be controlled at the site level. Global action is thus warranted. It can, of course, be changed whenever needed.

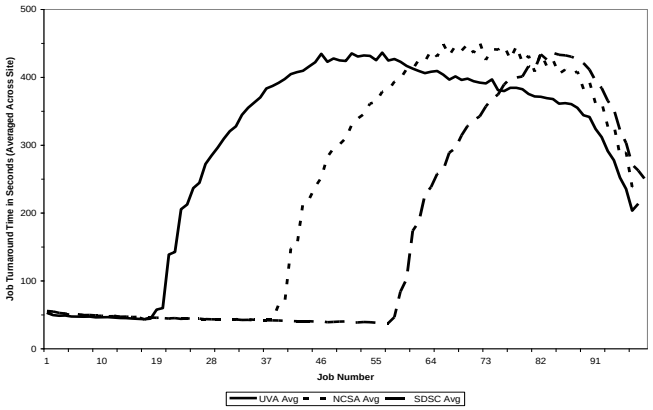

Figure 7. Node average job turnaround times for each site (Willow not running)

Figures 7 and 8 show the turnaround times for each job averaged across the five nodes at each site both without and with the Willow system operating. Again, we note that the effects of the disturbances were mitigated in a comprehensive way so as to ensure that the turnaround times for the jobs deemed the most important (those from the PHYS VO) were kept within reasonable bounds. We also note that the effect of the global policy change can be seen once again.

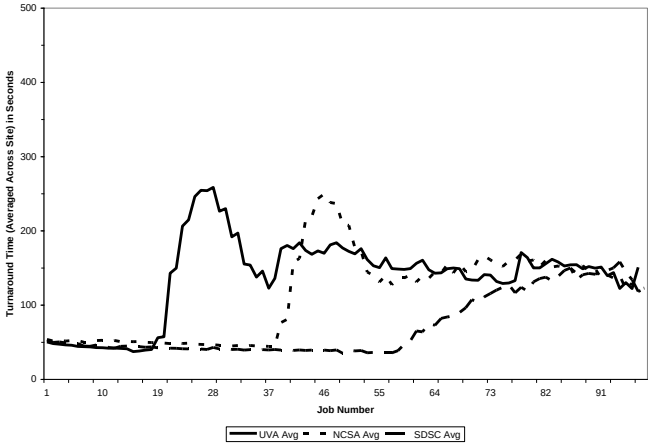

Figure 8. Node average job turnaround times for each site (with Willow)

V. DISCUSSION

In this paper, we have described the application of the Willow architecture to the monitoring and control of job turnaround time. The system ensured that a particular VOs high-priority jobs were kept running within a reasonable amount of time in the face of high CPU contention from other VOs.

It is important to note that the Willow architecture provides a general mechanism that can be used to enforce many different types of constraint on a distributed system. For example, the Willow architecture can be used to allow systems to respond to hardware failures such as a widespread loss of power in which various grid components become unavailable. Similarly, Willow can be used to facilitate response to security attacks by linking to sensors such as intrusion detection systems and activity monitors. TEDL specifications can be written to allow regional reaction to intrusions if that is all that is happening or a wide-area (perhaps national) reaction if an attack escalates.

Performance management with Willow, the subject of the study discussed above, faces a challenge that is often present in control systems. If changes are made to the controlled system to cope with a perturbation, it is quite possible for performance to return to a normal level or at least an acceptable level even with the perturbation present, as occurred in our study. But it is important to remove the changes made to accommodate the perturbation once the source of the perturbation has been removed. This requires some source of information about the perturbation and such information is not available from typical sensors.

Willow does not address this issue directly. The problem can be dealt with in a simple way by merely reverting to the default configuration after a predefined time period. If the perturbation is still present, then the system will react and apply an appropriate response. The downside to this approach is that the system might cycle through controlled and default states.

The GT4-based development grid that we used for our study was quite small, and thus raises the question of scale. How would Willow cope if it were deployed on a practical grid with perhaps thousands of nodes located at multiple host sites? We cannot answer this question directly because we do not have a large grid available for experimentation. The basic mechanisms that Willow uses for communication have been studied on larger grids and also on very large simulated grids [18][19]. The results suggest that the Willow system will scale to very large systems with no significant loss of performance.

VI. Conclusion

Today the management of grid systems requires manual coordination between administrators at participating sites. The problem with this approach is that it operates on human-time scales and does not facilitate the timely and consistent application of grid-wide actions needed to respond and adapt to changes in the environment.

The Willow architecture addresses this problem by providing an automated way to reconfigure grids in response to events either from the environment or from within the grid itself. The experiment in this paper demonstrated a proof-of-concept prototype for expressing and enforcing performance-oriented grid requirements. This represents an important milestone towards demonstrating the applicability of automated management architectures such as Willow for managing grid systems.

References

[1] TeraGrid. http://www.teragrid.org

[2] Open Science Grid. http://www.opensciencegrid.org

[3] J. Martin-Flatin, S. Znaty, J. Hubaux. "A Survey of Distributed Network and Systems Management Paradigms" Journal of Network and Systems Management. 7(1). pp 9-22. 1999

[4] M. Mountzia, D. Benech. "Communication Requirements and Technologies for Multi-Agent Management Systems." Eighth IFIP/IEEE International Workshop on Distributed Systems Operations and Management (DSOM'97). Sydney, Australia. 1997.

[5] M. Endler. "The Design of SAMPA." The Second International Workshop on Services in Distributed and Networked Environments. June 1995. IEEE Press.

[6] S. D. Taylor. "Distributed System Management Architectures". Masters Thesis, 1006, University of Waterloo, Ontario, Canada.

[7] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The Autonomic Computing Paradigm. Cluster Computing 9, 5–17, 2006

[8] J. Kephart and D. Chess. The Vision of Autonomic Computing. Computer Magazine, IEEE, January 2003.

[9] Hawkeye: A Monitoring and Management Tool for Distributed Systems. http://www.cs.wisc.edu/condor/hawkeye/

[10] SCALEA-G: A Unified Performance Monitoring and Analysis System for Grids. http://www.dps.uibk.ac.at/projects/scaleag/

[11] R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org/

[12] GridMon - Grid Network Performance Monitoring for UK e-Science. http://gridmon.dl.ac.uk/

[13] INCA Test Harness and Reporting Framework. http://inca.sdsc.edu/

[14] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre. MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. *CHEP 2004*, Interlaken, Switzerland, September 2004

[15] Virtual Data Toolkit. http://vdt.cs.wisc.edu/

[16] Pete Beckman, Suman Nadella, Nick Trebon, and Ivan Beschastnikh. SPRUCE: A System for Supporting Urgent High-Performance Computing. IFIP WoCo9 Conference Proceedings, Arizona July 2006.

[17] J. C. Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbu, and M. Gertz., "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications," Intrusion Tolerance Workshop, DSN-2002 The International Conference on Dependable Systems and Networks, June 2002.

[18] J. Rowanhill, "Survivability Management Architecture for Very Large Distributed Systems," Ph.D. Thesis, University of Virginia, July 2004.

[19] J. Rowanhill, "Efficient Hierarchic Management For Reconfiguration of Networked Information Systems," The International Conference on Dependable Systems and Networks, Florence, Italy, June 2004.

[20] P. Varner, "Policy Specification for Non-Local Fault Tolerance in Large Distributed Information Systems", M.S. Thesis, May 2003.