# OGSA-based Grid Workload Monitoring

Rui Zhang, Steve Moyle and Steve McKeever
Oxford University Computing Laboratory
Wolfson Building, Parks Road,
Oxford OX1 3QD, England
{ ruiz,sam,swm} @comlab.ox.ac.uk

Stephen Heisig
IBM T.J. Watson Research Centre
19 Skyline Drive, Hawthorne,
N.Y. 10532, USA
heisig@us.ibm.com

## Abstract

*In heterogeneous and dynamic distributed systems like the Grid, detailed monitoring of workload and its resulting system performance (e.g. response time) is required to facilitate performance diagnosis and adaptive performance tuning. In this paper, we present a workload monitoring infrastructure for this purpose. The infrastructure classifies and monitors workload across components in Grids based on the Open Grid Service Architecture (OGSA) in an end-to-end manner. It provides the abilities to assess what components are involved in processing a work unit, to report time elapsed at these components, and to capture concurrency and isolate which components are critical to overall performance observed. These are enclosed in an automatically constructed Response Time Service Petri Net (RTSPN) model. A tool is provided to accept queries about work units and visualise corresponding RTSPNs. The infrastructure is also designed and implemented so as to be portable, scalable and lightweight.*

## 1. Introduction

Computational Grids aim to deliver desired performance [18], typically in relation to response time. However, Grid applications normally contain a diverse set of work units with various importances, performance goals and invocation parameters, and run across a number of heterogeneous Grid components [21]. Different work units may dynamically pass through different (groups of) components as they are processed. This is especially true for Grids based on the Open Grid Service Architecture (OGSA) [17], whose Web Service features enable the implementation of higher-level Grid services via dynamic composition of lower-level services [17]. In order to deliver appropriate performance to Grid applications, workload must be monitored across Grid components between application end points in an end-to-end manner [17], assessing what components are involved in workload processing, how much time is spent on these components, their status, and how they contribute to overall response time. Such knowledge offers valuable assistance to performance diagnosis and tuning.

Nonetheless, little effort has been made to obtain the above information, especially the constitution of overall response time, which requires the modelling of concurrency in processing work units. In this paper, we present a workload monitoring infrastructure seeking to tackle these problems for Grids (OGSA-based Grids in particular) using response time as one performance metric. OGSA-based Grids are chosen mainly because OGSA is becoming a standard architecture for Grids and its Web Service features result in complex performance behaviours. More specifically, the infrastructure developed makes the following contributions:

- By classifying and tracing work units, the infrastructure discovers what Grid components are dynamically involved in the processing of different work units and reports how much time is spent at these components.
- A Response Time Service Petri Net (RTSPN) model is automatically constructed for every work unit using the collected data without knowledge of the application. The model is used to capture concurrency in processing the work unit and assess how overall response time is constituted. A visualisation based on the model is provided to the user/administrator.
- The instrumentation targets common OGSA-based Grid middleware and is readily available for various OGSA-based Grid deployments with minimal or no application code modification and configuration.

- An open standard monitoring methodology employed by the infrastructure is sufficiently general to be applied to other heterogeneous distributed systems with little adjustment.

The rest of this paper is structured as follows. Next section defines the RTSPN model. Section 3 describes the workload monitoring infrastructure. Experiments and results are presented in Section 4. Related works are reviewed in Section 5. The last section concludes and discusses future research.

## 2. OGSA-based Grid Performance

In this section, we decompose OGSA-based Grids into a hierarchy of components and model the run-time relations between them. The aim is to investigate the concurrency in workload processing and form a basis of the monitoring infrastructure discussed in next section.

### 2.1. Decomposition

In creating OGSA, efforts were made to encapsulate Grid resources and functions into Grid services – stateful Web Services with standard interfaces [17]. The implication is that an OGSA-based Grid can be decomposed into two levels of components – *Services* and *Platforms* – that process work units.

Services are higher-level components. An OGSA-based Grid can be viewed as a set of services (in OGSA specification, it is in actual fact service instances that are handling the work units, for simplicity however, we prefer the term service). A simplified eDiamond Grid [11], for example, consists of two services – an *image_retrieve* service and an *ogsa_dai* [16] service. In processing work units, services are invoked programmatically due to their Web Service features, resulting in dynamic influence on overall system performance.

Platforms are lower-level components. A service can be viewed as a sequence of platforms. When a Grid service is invoked, common platforms such as network and Grid container are invoked in a typical order. This is illustrated in Figure 3, where both the *image_retrieve* service and *ogsa_dai* [16] service is decomposed into three constituent platforms.

Following the above discussion, we claim that, in general, compared with platforms, services produce more dynamism and have a stronger impact on overall performance. Our discussion in the next subsection will focus on the service level.

### 2.2. Response Time Service Petri Net

The overall system performance in workload processing is determined by not only time elapsed on individual services but also how they compose overall response time. The former can be easily measured, whereas the later is decided by the concurrency in processing work units. For if two (or more) services are invoked sequentially, the overall response time will be the sum of time spent on each service, whereas if they are invoked in parallel (and are synchronised), it will add up to less. *Such concurrency must be accounted for before system performance behaviour can be truly understood.* It is determined by two types of relations between services – invocation relations and dependency relations. We define these two relations as:

DEFINITION 1. An *Invocation Relation* is a mathematical relation $\mapsto^* = \{\langle a,b\rangle | a \in S \wedge b \in S \wedge a \neq b\}$, where:

- $S$ is the set of services.
- $\langle a,b\rangle$ represents the invocation of service $b$ by service $a$ in processing a work unit.
- $a$ is called $b$'s *Parent Service* and $b$ is called $a$'s *Child Service*.

DEFINITION 2. A *Dependency Relation* is a mathematical relation $\Rightarrow^* = \{\langle a,b\rangle | a \in S \wedge b \in S \wedge a \neq b\}$, where:

- $S$ is the set of services.
- $\exists c \in S : \langle c,a\rangle \in \mapsto^* \wedge \langle c,b\rangle \in \mapsto^*$. That is, service $a$ and $b$ are invoked by the same service.
- $\langle a,b\rangle$ means service $a$ is dependent on service $b$. For example, $a$ takes $b$'s output as input.
- $b$ is termed $a$'s *Prerequisite Service* and $a$ is called $b$'s *Dependent Service*.

If a service invokes one or more child services, then it starts before all its child services start and terminates after all its child services terminate synchronously, while the child services may run in parallel or sequentially. If a service relies on one or more prerequisite services, then it starts after all its prerequisite services terminate synchronously.

Having defined the run-time relations between services, we opt to use a Petri-net (PN) to encode these relations and represent concurrency for a single work unit. The reasons are three-fold. Firstly, PNs are capable of modelling various aspects of concurrency including parallelism, sequence and synchronisation [13]. Secondly, theoretical results regarding PNs are plentiful: ranging from traditional properties such as boundedness and liveness [13] to deriving end-to-end response time from a generalised stochastic PN [14]. Finally, PN tokens map naturally to work units and thus PNs provide lively visualisation of how work units are processed in the Grid by firing transitions [13] (i.e. moving tokens from place to place that represents Grid component). We call such a PN *Response Time Service Petri Net (RTSPN)*. We assume (a) every work unit requires a response and (b) the services are invoked as soon as possible (i.e. once their prerequisite services terminate).

DEFINITION 3. A *RTSPN* is a Generalised *P-timed* Petri Net firing at maximum speed [13] and specified by a 4-tuple $(P, \Gamma, \Delta, W)$, where:

- $P$ is a finite set of places representing service fronts/rears, where a service front is the part of a service processing work unit requests and service rear is the part producing work unit responses.
- $\Gamma$ is a function from the set $P$ to the set of positive cardinal numbers, representing the time taken for a token to become available at places [13] and hence encoding the (processing) time a work unit spent at service fronts/rears.
- $\Delta$ is a finite set of transitions. Firing a transition, except for the source/sink transition representing entry/exit to the PN (the Grid), can bear one of the following four meanings:
  1. The simple returning of an invocation to a service that does not have child services (e.g. $T_1$ in Figure 1).
  2. The simultaneous invocation of a service's child services that do not have prerequisite services (e.g. $T_2$ in Figure 1).
  3. The synchronisation of a service's child services that do not have dependent services (e.g. $T_3$ in Figure 1).
  4. The synchronisation of a service's child services that have dependent services and in the mean time the simultaneous invocation of these dependent services (e.g. $T_4$ in Figure 1).
- $W \subseteq (P \times \Delta) \cup (\Delta \times P)$ is a finite set of arcs that connect transitions with related places.

An example is shown in Figure 1. The assumptions made for this particular example are, without loss of generality: B, C and D are invoked by A with C dependent on both B and D.

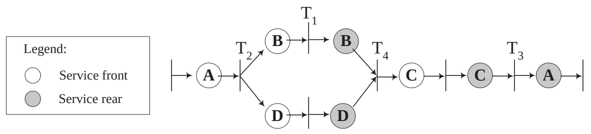

**Figure 1. Sample RTSPN**

Combining $P$ and $\Delta$, we can transform the RTSPN into a weighted directed graph, where the $P$ nodes are weighted with the corresponding value in $\Gamma$ and the $\Delta$ nodes are weighted $0$. The *longest* weighted path in the directed graph will then determine the overall response time seen by the work unit. We call this path the *Critical Path*, and the services on this path *critical*. Minor modification can be done to Dijkstra's Algorithm [10] to determine the critical path $C$. The complexity of the algorithm is $O(|P \bigcup \Delta|^2)$.

The overall response time is constituted by time elapsed at individual components according to:

$$t = \sum_{i=1}^{|C|} t_i = \sum_{i=1}^{|C|} (t_i^F + t_i^R) = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_i|} (t_{ij}^F + t_{ij}^R) \quad (1)$$

where $t_i$ is the time elapsed at $i^{th}$ service, $t_i^F$ and $t_i^R$ denote the elapsed time at service front and rear of the $i^{th}$ service. $t_{ij}^F$ and $t_{ij}^R$ hold similar meaning for the $j^{th}$ constituent platform of service $i$, $M_i$ is set of platforms composing service $i$.

Suppose, $t_A = t_{A^F} + t_{A^R} = 5, t_B = t_{B^F} + t_{B^R} = 10, t_C = t_{C^F} + t_{C^R} = 10, t_D = t_{D^F} + t_{D^R} = 5$ in seconds, then the overall response time will be $t = t_A + t_B + t_C = 25$ in seconds and service A, B and C will be the critical services.

In this section, we have presented a RTSPN model to capture service relations (and thus concurrency in workload processing).

## 3. Monitoring Infrastructure

In this section, we present the workload monitoring infrastructure. First the overall architecture is introduced, then three monitoring issues are discussed in detail.

### 3.1. Architecture

The infrastructure features a pipeline architecture with monitoring data flowing from bottom to top as depicted in Figure 2. Standard OGSA-based middleware, including the Globus [23] client, Tomcat [1], and

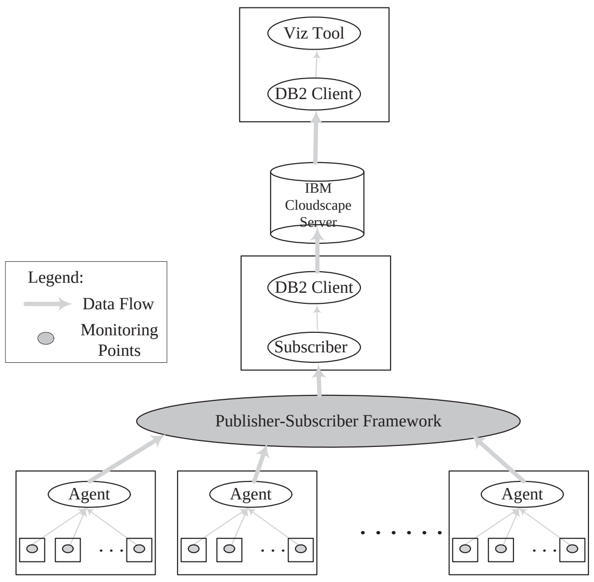

**Figure 2. Architecture**

Globus [23] are instrumented with *Monitoring Points (MPs)*, who reside in platform end points (e.g. Globus client and Tomcat are network platform end points) so that time elapsed at the platforms can be determined. Figure 3 illustrates an example instrumentation for a simplified eDiamond Grid [11] setting, where the (front-end) *image_retrieve* service delegates Grid-database-related tasks to the *ogsa_dai* [16] service. There are one or more platforms on each machine in the Grid and subsequently there are one or more MPs. There is an *Agent* on each machine, listening to the MPs for data, possibly batching them before publishing them to the publisher-subscriber [20] *Broker*. A client subscribes to the broker and asynchronously receives the data, which are in turn forwarded via a DB2 client to IBM Cloudscape and logged.

A unique ID is assigned to every work unit at its entry MP. The user or administrator is able to issue a query for the work unit's RTSPN, the output of the infrastructure. The query triggers the retrieval of relevant data via ODBC and DB2 client. A visualisation tool then uses the retrieved data to construct and display the corresponding RTSPN following Definition 3 in Section 2.

The hierarchical data reporting architecture we use moves the communication with the central broker away from the MPs and delegates it to local agents. As a result, it reduces overhead and improves scalability by providing the opportunity to batch up and compress local data before reporting them via a costly network transmission. The instrumentation is contained in common OGSA-based Grid middleware and compiles with the Application Response Measurement (ARM) [4] standard, enhancing portability.

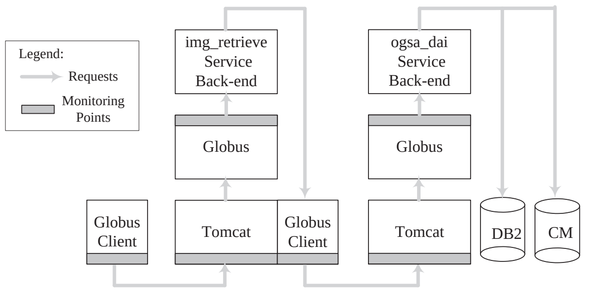

**Figure 3. Instrumentation for a simplified eDiamond Grid setting**

The MPs, whose algorithm is outlined in Table 1, complete three crucial monitoring tasks that are further detailed individually below.

**Table 1. Monitoring point algorithm**

```
If work unit request Then
    If first MP of work unit Then
        Classify work unit and generate a correlator;
    Else
        Receive correlator from previous MP and update it;
    Produce time stamp 1 and other measurements;
    Report measured data to agent;
    Send correlator to next MP;
If work unit response Then
    Produce time stamp 2 and other measurements;
    Report measured data to agent;
```

## 3.2. Classification

As they enter the instrumented Grid, work units are classified into service classes according to a combination of their properties (e.g. ownership (user), type, invocation parameters, etc.) and based on a pre-defined policy. The *Service class* represents the business importance and response time goal of a work unit [5]. The introduction of service classes in the monitoring infrastructure enables automatic Service Level Agreement (SLA) verification [21] and our future research on differentiated services [22] enforcement.

The service class is used to form the unique ID given to the work unit in the form of a 3-tuple <*agent, service_class, time_stamp*>.

### 3.3. Correlation

The work unit ID is communicated across MPs (deployed at services) so as to correlate data pertaining to the work unit after they are gathered.

Another important aspect of correlation is to derive the service relations. This is done automatically on-the-fly, *without knowledge of service logic or source code* which is generally not available. Deriving the invocation relation is straightforward – a service (like A in Figure 4) sends its service ID to every child service (B, C and D in Figure 4) it invokes. Deriving the dependency relation is more difficult. We know by Definition 2 that if a service (say A in Figure 4) maintains a list of latest terminated child services (say, B and D in Figure 4), these services could all be the prerequisite services of the next child service it invokes (C in Figure 4). By our assumption (b) for Definition 3, we are sure C is dependent on the last terminated service (B in this case), but for the rest (D in this case), the dependency is speculative. Currently, IDs of all the latest terminated child services are sent to the next child service (as in Figure 4). We are looking to make the speculation more precise by taking into account service specifications and/or multiple runs of the same type of work unit.

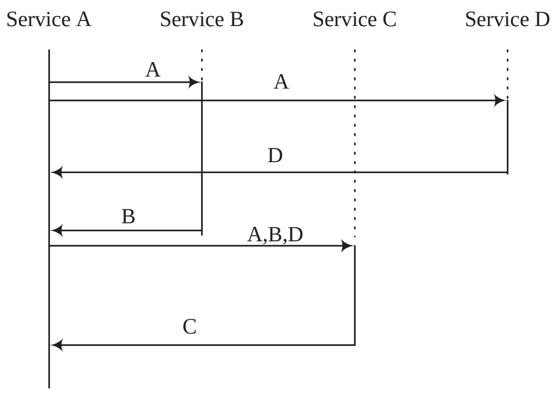

**Figure 4. Service correlation**

The above communication is actually conducted between the MPs instrumenting the Globus middleware of a service and the Globus client used to invoke its child services. Often since they reside in the same thread, this can be achieved via global variable sharing. Now (on the MPs) at each service, we have a *Service Pointer <parent_service_ID, current_service_ID>* encoding an invocation relation. Similarly, pointer *<current_service_ID, prerequisite_service_ID>* represents a dependency relation. These pointers will later be used by the visualisation tool to construct the RTSPN.

The above information, in addition to a counter of visited platforms in a service, is wrapped into a *Correlator* structure and floated along with the work unit request to other MPs.

### 3.4. Measurement

Measurements are taken at the MPs when a work unit travels by. The most important measurement produces two time stamps, one marking the moment when the work unit request passes by the MP, the other marking the moment when the corresponding response is produced. They are subtracted at the agent to calculate the local response time, $T$, which will be reported for calculation of time spent at services (and compositional platforms) using Equation 2 and 3:

$$t_i = |T_{i,0} - T_{i,N_i-1}| \tag{2}$$

$$t_{ij} = |T_{i,j} - T_{i,j+1}| \tag{3}$$

where $t_i$ and $t_{ij}$ were introduced in Equation 1, $i = 1, \ldots, N-1$, $j = 1, \ldots, N_i - 2$, with $N$ being the number of services and $N_i$ the number of MPs instrumenting service $i$. The subtraction eliminates clock difference between different MPs. This is crucial as the measurement is taken at distributed MPs where *clock synchronisation is usually not guaranteed*.

Also measured and reported to the agent is the correlator, the current component's ID, and potentially its resource usage status.

This section has presented the general architecture of the monitoring infrastructure, and addressed in detail the important monitoring issues of classification, correlation and measurement.

## 4. Experiment

In this section, we present results of applying the monitoring infrastructure to a simulated OGSA-based Grid setting. The hypothesis to be verified is that, for every work unit posted to the setting, the infrastructure is able to collect data and correctly construct a RTSPN.

### 4.1. Experimental Setting

We used a simulated setting that closely represents a possible eDiamond scenario. Integration of the infrastructure with eDiamond is already underway, and will be used to support our future research.

In the scenario, a radiologist tries to retrieve a list of mammograms assigned to him for analysis. As is illustrated in Figure 5, this involves six Grid services

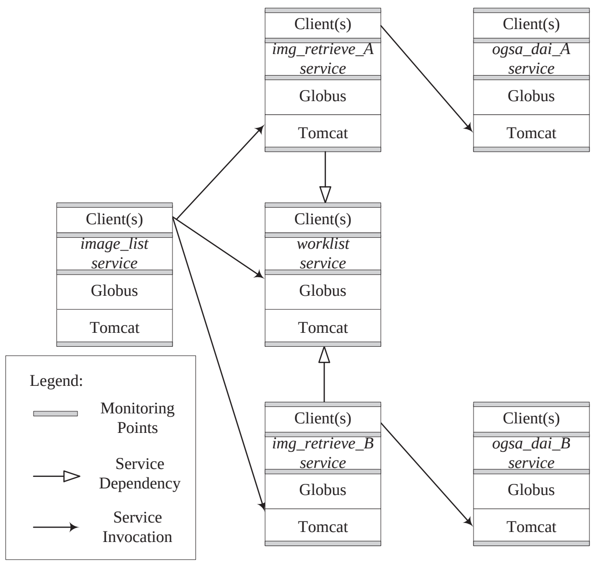

**Figure 5. eDiamond scenario**

each sitting on top of standard OGSA-based Grid middleware instrumented with MPs. The *image_list* service calls the *work_list* service asking for a list of IDs of the images assigned to the radiologist. Two image IDs are returned. Since a quick lookup shows they are stored in eDiamond site A and eDiamond site B respectively, the *image_list* service simultaneously issues two requests to the *image_retrieve* service on both sites. This leads to the invocation of the *ogsa_dai* service on both sites to obtain corresponding image URL from local database. Eventually, the URLs pointing to the two images are returned for the radiologist to retrieve the images via a separate channel. The above basically defines the invocation and dependency relations between the services shown in Figure 5. All the services are simulated and do nothing but cause a certain amount of delay and invoke other services.

### 4.2. Results

Table 2 shows a portion of the end-to-end monitoring data collected by the MPs for a certain work unit at a sample run of the experiment. It highlights data collected from MPs instrumenting service *image_list* and *image_retrieve_B* and omits the service pointers. The table shows the work unit was properly classified and monitoring data pertaining to it were correctly correlated together in an end-to-end manner. Table 2 accounts for what platforms and services were involved in processing the work unit and the amount of time spent on each of them. The elapsed time data are calculated using Equation 3. As opposed to a single overall response time, such a table provides details as to what happened along the way, and can be compared with historical data to detect abnormality.

**Table 2. End-to-end monitoring data**

| Platform | Hop | Service | Elapsed Time (Microseconds) |
|---|---|---|---|
| | ...... | | |
| Network | 0 | img_list | 807814 |
| Grid Container | 1 | img_list | 1198916 |
| Service Back-end | 2 | img_list | 1051049 |
| | ...... | | |
| Network | 0 | img_ret_B | 170362 |
| Grid Container | 1 | img_ret_B | 136051 |
| Service Back-end | 2 | img_ret_B | 2010015 |
| | ...... | | |

In order to construct the RTSPN, The data in Table 2 are further processed at the service level, deriving the time elapsed at each service according to Equation 2. It is presented along with the service pointers in Table 3 which describes the corresponding RTSPN. Upon query, a RTSPN, correctly reflecting the service relations (thus concurrency in workload processing) in Figure 5 is automatically constructed from Table 3 and returned as shown in Figure 6 (whose legend is the same as that in Figure 1). Only very basic visualisation is currently available, advanced visualisations such as highlighting the critical path, click-on display of properties associated with services and replacing each place with a clock graph representing elapsed time are being developed.

The RTSPN constructed presents a complete runtime performance snapshot for the work unit. Such a clear visualisation would be of great value to system administrators or even end users, who are often lost in the complexity the Grid exhibits. As detailed in Subsection 2.2, applying Dijkstra's Algorithm [10] to the directed graph corresponding to Figure 6 and using Equation 1, we have the overall response time is constituted as: $t = t_{image\_list} + t_{work\_list} + t_{image\_retrieve\_B} + t_{ogsa\_dai\_B} = 9433830$ in microseconds and service *image_list*, *work_list*, *image_retrieve_B* and *ogsa_dai_B* are the critical services. Clearly, likely delay causes can be quickly narrowed down to these critical services and their compositional platforms. Furthermore, the RTSPN's critical path property can potentially be exploited to present users with crucial information only and simplify workload visualisation in complicated scenarios where each work unit touches hundreds or even thousands of components. Assuming the same service class

**Table 3. RTSPN data**

| Service | Parent Service | Prerequisite Service | Elapsed Time (Microseconds) |
|---|---|---|---|
| img_list | Null | Null | 3057779 |
| img_ret_A | img_list | work_list | 1363543 |
| img_ret_B | img_list | work_list | 2316428 |
| ogsa_dai_A | img_ret_A | Null | 1441334 |
| ogsa_dai_B | img_ret_B | Null | 2742013 |
| work_list | img_list | Null | 1317610 |

of work units share a RTSPN (as they have similar or even identical properties) for a stable period, high frequency, high granularity monitoring can be focused on critical services to reduce overhead. In Figure 6, for example, turning off half of the MPs on non-critical services *image_retrieve_A* and *ogsa_dai_A* can reduce monitoring overhead by 17%. What is more, performance tuning middleware on these non-critical services may shift some resources allocated to this service class (of work units) to other classes without affecting overall response time of the former.

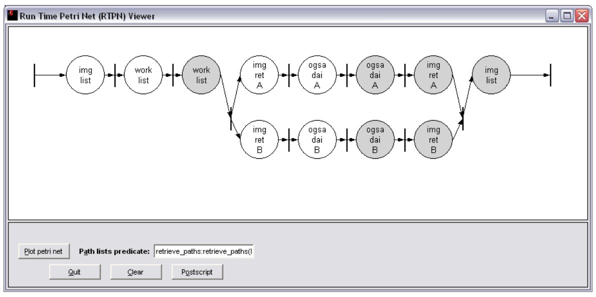

**Figure 6. Automatic RTSPN visualisation**

The infrastructure's overhead is measured to be only around 5.5 milliseconds per work unit per service on a linux server with four Intel Xeon (2.4GHz) CPUs. A major part of this overhead is caused by the standard Axis SOAP [17] parsing implementation in Java when the correlator (from Tomcat) is retrieved in the Axis context, and could be significantly reduced by replacing it with our own implementation that is better tailored for our needs.

## 5. Related Work

There are a number of existing infrastructures dedicated to Grid performance monitoring and analysis [24] [6] [7] [8]. Most focus on monitoring architecture, for instance, implementing the GGF Grid Monitoring Architecture [15]. However, little has been done to correlate collected data and offer end-to-end performance monitoring of work units, which, we have achieved and argue is fundamental in aiding performance diagnosis and tuning. Furthermore, none of the above work targets OGSA-based Grids such as eDiamond [11], Astro Grid [2] and service-based Data Grid [3].

There are other more general distributed system monitoring approaches such as Pinpoint [12], Magpie [9] and Netlogger [19] which do provide end-to-end tracing of work units. However, they have drawbacks if applied in the Grid paradigm. Magpie [9] does not explicitly float a correlator along with the work unit but relies on more heavyweight methods to correlate data pertaining to a work unit. This requires prior knowledge about the application and is at risk of being overwhelmed by data in a large distributed environment like the Grid. Mapgie's monitoring mechanism relies heavily on on Windows event tracing functions and undermines its portability. Pinpoint [12] only addresses isolation of root causes of failure and has yet to considered performance. Netlogger [19] assumes synchronised clocks which is itself an outstanding nontrivial challenge to the Grid world. Moreover, the tricky task of instrumentation and data correlation is left to the users. Our approach aims to address these gaps.

Additionally, research discussed above has not featured any model accounting for the concurrency in workload processing. As a consequence, critical insights like how the overall response time observed by a work unit is constituted remain unclear. Our approach, on the other hand, has made such an attempt in RTSPN.

## 6. Conclusion and Future Work

In this paper, we have presented a workload monitoring infrastructure for OGSA-based Grids. In our approach, work units are classified and traced end-to-end as they travel through the Grid, with time elapsed at sojourned components and their status reported. At the heart of the infrastructure is a RTSPN model that captures concurrency in processing a work unit and isolates components critical to overall response time. The model is constructed automatically from data collected for a work unit and presented using a visualisation tool to give Grid users and administrators useful insights into system performance behaviour.

To aid portability and minimise application code changes, instrumentation in the infrastructure prototype targets common OGSA-based Grid middleware and follows the ARM standard. A distributed hierarchical architecture and data batching mechanisms are in place to ensure scalability and reduce network flows.

The presence of the monitoring infrastructure has enabled future research on some promising topics in relation to Grid performance:

- *End-to-end response time tuning:* The aim is to provide differentiated services and optimise system goals by proposing adaptive closed-loop tuning mechanisms based on the monitoring infrastructure.
- *RTSPN mining:* Data mining and machine learning techniques can be applied to individual RTSPNs to detect patterns (paths, component and requests properties) causing delays. RTSPNs can also potentially be merged into a single stochastic net.
- *Adaptive monitoring:* Critical services revealed in RTSPNs and delay cause analysis can be used to dynamically adjust monitoring granularity and frequency to reduce overhead.
- *Inter-domain correlation:* Since work units may transcend Grid domains with distinct policies, a common correlation context needs to be architected to perform classification mapping at domain borders so that work units can be traced across these domains.

## 7. Acknowledgement

Rui Zhang and Steve Moyle are supported by the UK Department of Trade and Industry project (Grant THBB/C/008/00025): Heterogeneous Workload Management and Grid Integration. We are also grateful to the eDiamond project for their advice on OGSA-based Grid issues.

## References

[1] Apache jakarta tomcat project. http://jakarta.apache.org/tomcat/.

[2] The astro grid project. http://www.astrogrid.org/.

[3] Multi-user multi-job resource utilisation. http://www.lesc.ic.ac.uk/projects/optimise.html.

[4] Application response measurement (arm) – issue 4.0 java binding. Technical report, The Open Group, 2003.

[5] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2), 1997.

[6] M. Baker and G. Smith. Gridrm: A resource monitoring architecture for the grid. *Lecture Notes in Computer Science*, pages 2536–2680, 2002.

[7] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda. From cluster monitoring to grid monitoring based on grm. *Proceedings of 7th EuroPar2001 Parallel Processings, Manchester, UK*, pages 874–881, 2001.

[8] B. Balis, M. Bubak, W. Funika, T. Szepieniec, and R. Wismuller. An infrastructure for grid application monitoring. *Lecture Notes in Computer Science*, 2002.

[9] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'04),*, December 2004.

[10] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[11] J. M. Brady, D. J. Gavaghan, A. C. Simpson, M. Mulet-Parada, and R. P. Highnam. eDiaMoND: A grid-enabled federated database of annotated mammograms. In F. Berman, G. C. Fox, and A. J. G. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 923–943. Wiley Series, 2003.

[12] M. Chen, E. Kiciman, E. Brewer, and A. Fox. Pinpoint: Problem determination in large, dynamic internet services. *Proc of DSN*, 2002.

[13] R. David and H. Alla. *Petri Net and Grafcet: Tools for modelling descrete event systems*. Prentice Hall, 1992.

[14] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt. Response time densities in generalised stochastic petri net models. In *WOSP '02: Proceedings of the third international workshop on Software and performance*, pages 46–54. ACM Press, 2002.

[15] B. T. et. al. A grid monitoring architecture. *http://wwwdidc.lbl.gov/GGFPERF/GMA-WG/papers/GWD-GP-16-2.pdf*, 2002.

[16] M. A. et. al. Ogsa-dai: Two years on, 2002. http://www.nesc.ac.uk/events/GGF10-DA/programme/papers/15-Antonioletti-OGSA-DAI-DA-WS-final.pdf.

[17] I. Foster and C. Kesselman. *The Grid 2: Buleprint for a New Computing Infrastructuer*. Morgan Kaufmann, 2004.

[18] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[19] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. *Proceedings of the IEEE Mascots 2000 Conference (Mascots 2000), LBNL-46269*, August 2000.

[20] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. Java messaging service specification. Technical report, Sun Microsystems, 2002.

[21] S. Heisig. Treemap for workload visualization. *IEEE Computer Graphics and Applications*, 23, 2003.

[22] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated service fields (ds field) in the ipv4 and ipv6 headers. RFC 2474, Internet Engineering Task Force, 1998.

[23] T. Sandholm and J. Gawor. Globus toolkit 3 core c a grid service container framework, 2003. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf.

[24] H.-L. Truong and T. Fahringer. Scalea-g: a unified monitoring and performance analysis system for the grid. *Proceeding of 2nd European Across Grids Conference, Nicosia, Cyprus, Jan 28-30*, 2004.