Proceedings of the UK e-Science All Hands Meeting 2004, © EPSRC Sept 2004, ISBN 1-904425-21-6

Performance Architecture within ICENI

Stephen McGough, Laurie Young, Ali Afzal, Steven Newhouse and John Darlington

London e-Science Centre, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
Email: lesc-staff@doc.ic.ac.uk

Abstract. This paper describes the architecture built into the Imperial College e-Science Infrastructure (ICENI) for handling performance meta-data. The architecture provides a means to gathering performance information, processing this information to populate the performance store, and to use this performance information to aid in the selection of resources and component implementations. The Performance Framework is developed in a "pluggable" manner allowing alternate implementations of the three main features to be used. Performance Stores may be either data stores or based on analytical models.

1 Introduction

Grids, federations of distributed computing, storage and software resources owned by different organisations, are beginning to emerge in industry, commerce, research and academia. Expectations within these communities as to the quality and behaviour of the Grid are very diverse: some users expect resources to be provided for free with 'best effort' quality of service while others are happy to pay per use but expect defined service level agreements in return. Delivering such discriminating levels of service, possibly in return for payment, involves the integration of the Grid's underlying fabric (the resources), with the middleware that exposes this capability and with a user environment that is able to expresses these requirements. Our focus in this paper is to show how bringing together elements from the fabric (reservations) and middleware services (performance & scheduling), with the user and their applications (workflow, application performance models & execution constraints) will allow us to build a Grid infrastructure that will deliver a predictable user experience.

An integrated approach to building Grid middleware has been a central philosophy within the ICENI (Imperial College e-Science Networked Infrastructure) activities at the London e-Science Centre [5]. We have been using ICENI to prototype a Grid infrastructure that will provide a predictable user experience through the co-ordinated use of (initially): reservation enabled compute resources; an ability to capture and record performance information obtained during application execution; the ability to define the workflow and provide performance annotations when constructing an application from components and a scheduling system capable of exploiting this collected meta-data to optimise the placement of the application to reduce overall execution time. This enables us to ensure that appropriate reservations are obtained on the relevant compute resources and to ensure a predictable execution time.

ICENI is a component based end-to-end Grid infrastructure that provides a mechanism for defining workflows in terms of components. These components are scheduled, through the ICENI scheduling system which determines the "best" mapping of implementations of each component onto a subset of available resources through the goals of the various stakeholders i.e. users, resource providers and managers of the virtual organisation. In order to efficiently map components onto resources it is necessary to have some knowledge of the execution time for each component.

We have developed a performance repository system within ICENI which is capable of monitoring running applications to obtain performance data for the components within the application. This data is stored within the performance repository with meta-data about the resource the component was executed upon, the implementation of the component used and the number of other components concurrently running on the same resource. There is also provision for the component implementation designer to define other meta-data that should be stored. This could include such things as the problem characteristics which will affect the execution time.

In future runs of applications through the ICENI system the performance data stored can be used by the scheduler to estimate the execution times for each component within the workflow and hence the overall execution time of the application. As the meta-data stored about each component's execution time includes data about the number of components concurrently running on a resource it is possible for the scheduler to compute the execution time for the

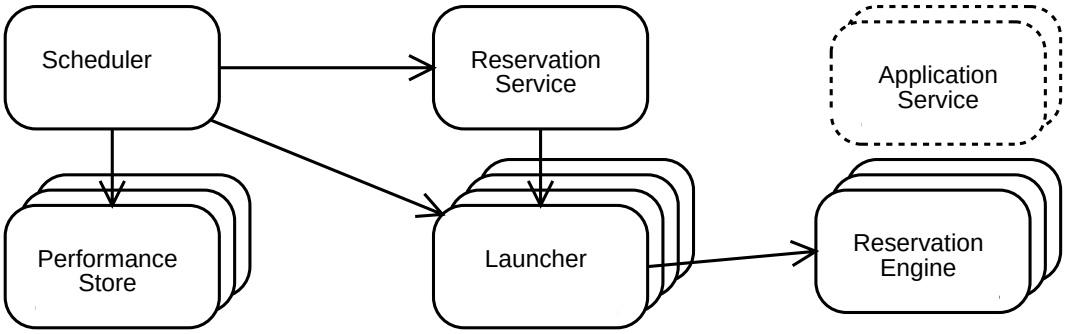

Fig. 1: The Trinity architecture: Scheduling, Reservation and Performance Prediction

whole application given that components can be co-allocated to a resource.

2 Implementation Architecture

We beleave that there is a trinity between the scheduling of applications, the ability to reserve resources and the ability to accurately predict the performance of the components constituting the application. Without performance modelling one cannot extract a critical path, nor produce accurate reservations, and without reservations one cannot control the predictability of execution times. In the rest of the paper we outline an architecture for using this trinity to efficiently deploy applications over a Grid. We further show how this architecture has been integrated into the ICENI[5] Grid middleware.

The basic architecture is illustrated in Figure 1. A Scheduler may interact with multiple Launchers where each Launcher represents one or more resources. Each Launcher may be associated with one Reservation Engine, if reservations are possible. The Scheduler interacts with the Reservation Service which in turn communicates with the Reservation Engines through the Launchers. There may be multiple Performance Stores each of which can be interrogated by the Scheduler. Once a workflow is instantiated it will have an Application Service which exists until the workflow terminates.

In this architecture the Scheduler acts as the controlling entity with workflows being submitted to it. Once the Scheduler has received the workflow it can determine the critical path by determining the execution times of the component activities through requests to the Performance Stores. The scheduler can then produce (potentially more than one) concrete workflow using a subset of the resources available within the Grid. These concrete workflows can be passed to the Reservation Service to generate reservations. The Reservation Service will talk to all relevant Launchers and make requests to their Reservation Engines if available. The Scheduler will communicate with each Launcher involved in the selected concrete workflow to initiate the appropriate parts of the workflow. If components in the workflow are to be enacted at a later point due to a reservation they are held in the Application Service until they are due to be launched.

3 Scheduling Applications using Workflow and Performance data

Various scheduling algorithms can be used within ICENI by using the 'pluggable' Scheduling Framework. Many schedulers have been written for the ICENI framework, some of which are capable of scheduling workflows in terms of the execution times of the components. The scheduler can communicate with the Performance Repository through the Scheduling Framework. The scheduler requests execution times given implementations, resources and co-allocation counts along with other meta-data defined by the component implementation designer. This information can be combined into the workflow to enable the scheduler to determine the overall execution time for the application and determining the critical path. The predicted execution times of each component in an application allows advanced reservations to be made for that execution, when this is allowed by the low level execution environment. Delayed enactment of components can also be performed.

Each Grid application is described as an abstract workflow, a description of activities to be carried out and the dependencies between them. The task of a Scheduler is to allocate each task to a Grid enabled resource, thus creating a concrete workflow. A number of algorithms are available to perform this task, such as *simulated annealing*, *complete information game theory*, *best of N random* and *exhaustive search*[13]. Each of these algorithms at some point has to evaluate and contrast potential schedules. When optimising for time (as in most cases)

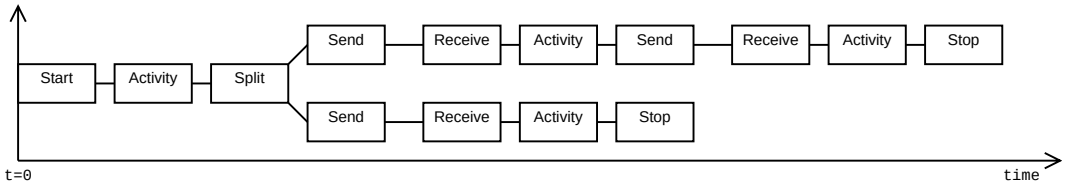

Fig. 2: Scheduling Applications based on start time

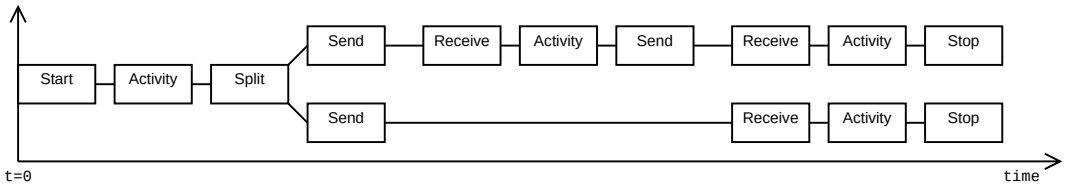

Fig. 3: Scheduling applications based on finish time

this is done by computing the expected execution time.

In order to compute the expected execution time the critical path though the application is computed. This involves computing the start and end time of each activity. This information is stored and later passed on to the reservation service. A number of the activities will have no dependencies on any task and are thus considered to be 'starting' activities, which are assumed to start at time $t = 0$ without loss of generality. A predicted duration is then obtained by querying the Performance Repository, taking into account the number of activities the scheduler has placed on the same resource, the resource specification, the implementation selected and other meta–data defined by the component implementation designer. The expected end time of the activity is used as the start time of the next dependent activity. This is repeated until all activities have been processed (see fig 2). This gives the earliest time an activity can start. The latest end time is then the duration of the application. The scheduler can then select concrete workflows with the shortest application duration.

Activities which do not lie on the critical path have much more freedom in terms of when they execute. This results in more freedom for the reservation engine in making reservations. This is calculated by using the same procedure as above, except: all 'stop' activities (those with no activities dependent on them) are considered to occur simultaneously at the end of the application; and the analysis is done backwards (see fig 3). This gives the latest time an activity can start without increasing the total application duration. If the start and end times for an activity match when computed in both directions then the activity lies on the critical path.

4 Gathering and Using Performance Data

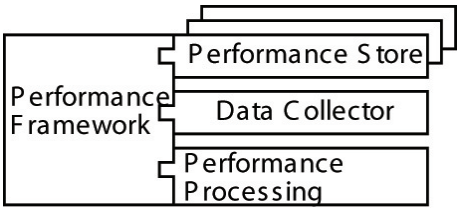

Fig. 4: The Performance Repository Framework

The Performance Repository has been designed with the capability to monitor running applications, to obtain performance data for the components within the application and to store new data when available. This is provided through the 'pluggable' Performance Repository Framework (Figure 4). The data is stored within the Performance Store with meta–data about the resource the component was executed upon, the implementation of the component used and the number of other components concurrently running on the same resource. There is also provision for the component implementation designer to define other meta-data that should be stored. This could include such things as the problem characteristics which will affect the execution time. For example in the case of a component which solves a set of linear equations the component developer can indicate that the number of unknowns will have an effect on the performance of the component. This can be added to the meta-data about the component. When the application is run the number of unknowns will be recorded and when the Perfor-

mance Repository is queried if a value for the unknowns is given then the returned value will be derived using this value.

When requesting data from the Performance Repository all attached performance stores will be asked for their data. This data is taged with metadata indicating the confidence the store has on the provided data. This is used to provide weightings on the data as it is combined. When obtaining values from the Performance Repository the client may specify a value x indicating the proportion of "extra" time that should be added to the return value. Values are generated as the mean of the recorded values. Thus the return value generated from the store from a set of records with mean $\overline{d}$ and standard deviation σ the return value is computed as

$$\overline{d} + x\sigma.$$

4.1 Collection of Performance Results

Each component within ICENI's extended component programming model may have multiple input/output ports which will lead to an activity being performed either between the component starting up and some port firing, or between ports firing on the same component. This implies that each component may have multiple activities. These activities cannot be transparently observed by the ICENI system. However, the port firing (and component start, end) times are recorded by ICENI allowing the activity times to be recovered along with any data and meta-data provided by the component implementation designer.

Each time a component starts up or a component port fires an ICENI event message is triggered. The performance repository listens for these events and records them for each running application. Once the application has finished executing the raw timings and meta-data provided by the component implementation designer are processed to compute the activity times and stored permanently within the performance repository.

4.2 Storing Performance Data

The Performance Repository allows multiple storage implementations to be used through a standard interface. It is assumed that all performance stores will provide persistent data storage. Multiple performance stores can be provided from a single Performance Repository and multiple Performance Repositories may exist within the same ICENI space. If multiple stores are available results will be aggregated between the stores based on some notion of the quality of the data provided by the store. As the performance store provides information given some specification of implementation and resource (amongst other data) it is possible to provide an analytical model which provides information in place of a store. If multiple stores are attached to a performance repository each store has the ability to add new activity information once it is collected.

Currently ICENI contains performance stores for a serialised object store and analytical models for a range of simple components. A database store is currently under development.

In general the store will not be capable of providing predicted execution times for all possible combinations of resource, component and problem specific descriptions. The performance repository is capable of using regression techniques to provide estimates in cases where previous data fails to provide an estimate. This is currently an ongoing research topic within the group.

5 Making Advanced Reservations

The Reservations Service attempts to co-allocate all the resources required for the execution of the workflow. It does so by entering into negotiation with the appropriate resource manager, abstracted through the Reservations Engine, running on each resource in order to make advanced reservations for the time and duration specified in the concrete workflow. The negotiation is bound by the constraints defined by the earliest and latest start times for an activity as calculated by the scheduler. The negotiation protocol is based on WS-Agreement[9].

Advanced reservations can, obviously, only be created if they are supported by the underlying resource manager, otherwise the scheduler has to resort to a best-effort service with regards to any unreservable resources. If the negotiation fails, i.e. the resource cannot create a reservation satisfying all of the application's constraints, then the reservations service can consider alternative schedules.

The Reservation Engine exposes the advanced reservation capability, if it exists, in the underlying resource management system. We are using Sun Grid Engine to support our advanced reservation work. Until advanced reservation support is available we are modifying the access control lists for the required resources to only allow a specific job from a specific user to gain access to the resource at the specified reservation time.

With this interim solution, reservation requests that have been accepted are added into a queue and activated at their designated start times. Any jobs still running on the resource are terminated and the access control lists associated with the resource are

modified to allow access only to the user who has requested the reservation. Henceforth, only that user can now submit jobs to the resource. When the reservation end time is reached, the access control lists are restored to their default configuration. Any jobs still running on the resource, at the expiry of the reservation will either be allowed to continue or terminated based on the local resource policy.

The Reservations Engine handles all incoming requests for reservations. These requests are validated in terms of the user access permissions and the availability of the named resource. Once the request is validated, the Reservation Engine checks for conflicts with existing reservations. A conflict occurs when the time frame for which a reservation is requested overlaps that of another reservation already created on the same resource. If the reservation request is valid and there are no conflicts, the Reservation Engine creates a reservation on the resource. If a conflict is detected, the reservation request is rejected and the client is notified. The Reservations Engine can also potentially return a list of alternative reservations: reservations on the same resource, for the same duration of time, but beginning at a different time. The client can then choose to create a reservation from the list of alternatives presented to it or abandon the negotiation process.

6 Related Work

Scheduling with Advanced Reservations has been the subject of few studies. Even fewer investigate the performance issues surrounding advance reservations. Nevertheless, in this section we attempt to draw parallels and identify the differences between our approach and existing work.

Advance reservations schedulers tend to use the maximum predicted execution time or, at best, a conservative estimate of the execution time of a workflow, or activity within a workflow, when creating reservations on a resource [10, 6, 12]. Our approach uses the mean predicted execution time of a component plus a multiple of the standard deviation. The problem with the former approach is that there may be quite a significant difference between the mean and maximum predicted execution times, especially in the case where the performance predictions were obtained using theoretical techniques, e.g. structural analysis of programs. In this case, using maximum predicted execution times could unnecessarily delay other applications, even though the running application may finish execution well before the expiry of it's reservation.

Using the latter model, predicted execution times take into account a big percentage of previous runs of the activity, disregarding spurious and widely varying results. This results in a more accurate prediction, with a very high probability that the activity will finish execution within the reservation time. Thus, for a minimal loss in reliability as compared to the case where maximum predicted run-times are used, we have eliminated the problem where reservations for other applications/activities on the same resource could be unnecessarily delayed. Since reservations are now being requested for a shorter time as compared to the maximum predicted run-time, the likelihood that a reservation request will be rejected is reduced (see [8]). Thus, we have also reduced the mean difference between requested reservation times and actual reservation times.

Our performance model for workflows using reservations assumes that batch jobs and jobs waiting in the queue have lower priority than reserved jobs. Our main concern is reliable and efficient execution of a workflow by reservation of resources. Jobs submitted outside of reservations are only executed when the resource is not reserved and executing the job would not effect any reservation in any way, i.e. Backfilling. Hence, the performance results with regards to mean wait-times and mean difference between requested and actual reservation times as documented in [10] do not apply in our case.

The GRADS project [11, 7] uses the approach of fine grain monitoring of applications which they have found to affect the performance of the application. Or approach is much more course-grained thus not suffering from this problem.

Our model for the reliable execution of workflows holds regardless of scheduling algorithm and backfilling. Analysis of the case where the reserved jobs have lesser priority than jobs in the queue is the subject of further work.

Foster [4] proposed a technique for making advanced reservations in GARA. This work has been supplanted by the WS-Agreement which we use in this work. Our work matches the architectural design currently being discussed through the Open Grid Services Architecture (OGSA) [2] Execution Management Service (EMS) subgroup within the Global Grid Forum (GGF) [3]. The trinity we have identified should also be present in any EMS enabled system. Our work also fits within the scope of the work undertaken within the EU-funded APART project [1].

7 Conclusion

In this paper we have described the performance architecture placed within ICENI to provide what we have termed the "trinity" of scheduling services;

scheduling, reservation and performance services. This allows ICENI to provide much better predictions on the overall execution times of applications composed of individual and interacting components. To provide such control a system requires a performance modelling mechanism that allows both application structure and empirical performance results to be recorded, stored, retrieved and analysed. The application structure is necessary for the reservation mechanism, the empirical data is needed to reduce performance overheads. Increasing numbers of resource management systems can support advanced reservations, and together with schedulers to exploit them we have a trinity of performance analysis, reservation mechanism, and scheduling systems that can provide the required predictability. This capability has been fully integrated into the ICENI grid middleware system.

ICENI is distributed under a SISSL-like open source license, and has a flexible and extensible architecture that allows new services and new implementations (at each stage of the performance driven scheduling process) to be plugged in. This allows ICENI to be used as a framework to explore these performance scheduling issues with a Grid support e-science applications.

References

1. APART: Automatic Performance Analysis: Real Tools. `http://www.fz-juelich.de/apart/`.
2. Open Grid Services Architecture. `https://forge.gridforum.org/projects/ogsa-wg`.
3. Global Grid Forum. `http://www.ggf.org`.
4. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, 1999.
5. N. Furmento, W. Lee, A. Mayer, S. Newhouse, and J. Darlington. ICENI: An Open Grid Service Architecture Implemented with Jini. In *SuperComputing 2002*, Baltimore, USA, November 2002.
6. Maui Scheduler. http://www.supercluster.org/maui.
7. C. Mendes and D. Reed. Monitoring large systems via statistical sampling, 2002.
8. Rui Min and Muthucumara Maheswaran. *Scheduling Co-Reservations with Priorities in Grid Computing Systems*, pages 266–268. 2002.
9. Grid Resource Allocation Agreement Protocol. `https://forge.gridforum.org/projects/graap-wg`.
10. Warren Smith, Ian Foster, and Valerie Taylor. *Scheduling with Advanced Reservations*, pages 127–132. 2000.
11. Fredrik Vraalsen, Ruth A. Aydt, Celso L. Mendes, and Daniel A. Reed. Performance contracts: Predicting and monitoring grid application behavior. In *GRID*, pages 154–165, 2001.
12. Lingyun Yang, Jennifer M. Schopf, and Ian Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of SuperComputing 2003*, pages 1–16, 2003.
13. L. Young, S. McGough, S. Newhouse, and J. Darlington. Scheduling Architecture and Algorithms within ICENI. In *UK e-Science All Hands Meeting*, August 2003.