Project Number: GXS 1101

Extension of Grid Portal Functionalities with Collection and Visualization of Usage Statistics

A Major Qualifying Project Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the
Degree of Bachelor of Science by

Alessandra Anderson

Sam Moniz

April 28, 2011

Professor Gábor N. Sárközy, Major Advisor

Professor Stanley M. Selkow, Co-Advisor

Abstract

The WS-PGRADE Grid Portal allows users to create and maintain workflows through an intuitive user interface. However the current version lacks the ability to share metrics about the system. To provide these metrics a new portlet, database and web service were developed. The service is responsible for collecting and storing metrics in the database and the portlet is responsible for display of these metrics. These additions enable end-users to retrieve statistics on the portal, user, DCI's, resources, concrete workflows, workflow instances, and individual jobs from the workflow graph.

Acknowledgements

First of all we would like to thank our sponsor, MTA SZTAKI and the Laboratory of Parallel and Distributed Systems (LPDS) and the laboratory head Professor Dr. Péter Kacsuk for allowing us to have an opportunity to work with the LPDS staff to create an interesting project dealing with the WS-PGRADE Grid Portal. Secondly we would like to thank Worcester Polytechnic Institute for allowing us this great experience to travel abroad for our capstone project.

The following individuals deserve particular acknowledgement for their contributions to our project and for always making us feel welcome and a part of the LPDS community. As mentioned previously, Professor Dr. Péter Kacsuk who provided us with the opportunity to work in Budapest with LPDS branch. Also we would like to thank Dr. Miklós Kozlovszky for his help and support throughout the project, making us feel welcome, checking up on us when we were ill, answering our daily questions, and always ensuring our time here was both enjoyable and productive. We would like to thank Sándor Ács for his help throughout the project as well as beneficial suggestions and ideas on what to do in Budapest as well as Gábor Herman for his friendly approach and assistance with the testing phase of our project. Furthermore we would like to thank Ákos Balaskó for all his technical support, ideas, and being there to answer daily questions and sort through bugs as well as Krisztián Karóczkai for his support with the database and setting up our development environment.

We would also like to thank Kitti Varga who helped us daily with printing, ordering food and suggesting social events in Budapest as well as her welcoming attitude towards us in the office. We would like to thank Réka Makkos who assisted us with the language barrier, finding train schedules, providing comfort and always checking in to make sure we were alright. Furthermore we would like to thank Zsófia Jávor, who would let us know whenever anything was going on and Dr. Róbert Lovas who would always take the time to have a friendly conversation. And to everyone on the staff of LPDS, thank you for providing a warm environment and making our time here both enjoyable and comfortable, we really enjoyed our stay. Finally we would like to thank our advisor Gábor Sárközy and co-advisor

Stanley Selkow for their guidance on our project, the preparation that went into our being here, and our stay in Budapest. We would like to especially thank Professor Sárközy for advising this project and always making sure we were on the right track, both for our project and for our experiences in Hungary.

Table of Contents

Table of Figures

1: Background

In the field of scientific computing there are some complex computational problems that require a large amount of resources to solve. Such tasks as parameter studies, analysis, and other complicated problems are difficult to accomplish due to lack of resources or computational power. One of the solutions to these problems is grid computing.

Grid computing is used to share tasks over multiple computers and shared resources. MTA-SZTAKI, located in Budapest, Hungary, has developed The WS-PGRADE Grid Portal which is a web based, service rich environment for the development, execution and monitoring of workflows and workflow based parameter studies on various grid platforms. The WS-PGRADE Grid Portal uses high-level graphical interfaces to allow all levels of users to submit applications, in the form of directed acyclic graphs (DAG), to a large variety of grid solutions. The DAG defines the dependencies between components of the user's workflow and the job manager then uses various Grid resources for processing the application. Furthermore the portal can access multiple grids simultaneously which allows easy distribution on multiple platforms [10].

The portal allows users to run jobs on multiple grid infrastructures such as gLite and other middleware as well as local clusters [9]. Furthermore they can submit a workflow to multiple Distributed Computing Infrastructures (DCI) which are each comprised of numerous resources.

1.1: Project Statement

The objective of this project was to integrate a new service into the WS-P-GRADE Grid Portal which would collect, store and present data about the execution of jobs and workflows on the WS-PGRADE Grid Portal. This addition allows end-users, communities, and administrators to retrieve statistics on the portal usage.

The design of our project had three major components: data collection, metric calculation and visualization. The goal of our data collection component was to receive data from the WS-PGRADE Grid Portal and reduce it to an efficient structure. Our metric

calculation component consumed that data and calculated the portal's statistics. Finally our visualization component displayed the statistics to the user in a meaningful form.

The motive behind this project was to provide a new service in the WS-PGRADE Grid Portal that would be a useful addition. Although this project is mostly to provide a new feature for the users it is also helpful for administrators. For administrators this feature will allow them to track of the load on different aspects the portal, as well as be able to monitor different levels of usage so they can better provide for the user. For the user our service provides feedback on the execution of their jobs and workflows.

1.2: Grid Computing

Grid Computing was originally proposed as a global system to solve computationally intensive problems that could be solved in a reasonable amount of time even with state of the art supercomputing resources[6].This problem was solved by aggregating multiple computing resources that may be geographically or architecturally distinct.

On top of these resources there is a grid middleware layer that hides the low level hardware and software differences between resources and provides a standardized interface for use. To add another layer of abstraction, it is also possible to use a grid portal to hide the differences between multiple grid middlewares, such as the WS P-Grade Portal developed by MTA SZTAKI's Laboratory of Parallel and Distributed Systems.

There are two main categories of resources used in grid computing. First are dedicated resources called service grids. These can be single monolithic machines or they can be computing clusters. The primary benefit of these resources is that they are dedicated, trustworthy and powerful. The other type of resource is commonly referred to as a desktop grid. These primarily function using a concept called *cycle scavenging* where owners donate their unused CPU time to work on a problem farmed out to the grid[2]. The considerations of desktop grid systems are different than those of service grids as there are not the same guarantees of availability and trust that there are with service grids [13].

1.2.1: Workflows and Jobs

One of the advantages of the distributed computing paradigm of grid computing is the capability for parallelization. This is further suggested by the structure of the applications or workflows created to be executed on such grid systems. At a high level a

workflow is defined by a Directed Acyclic Graph (DAG) for which the nodes are jobs and the edges are inputs and outputs of those jobs.

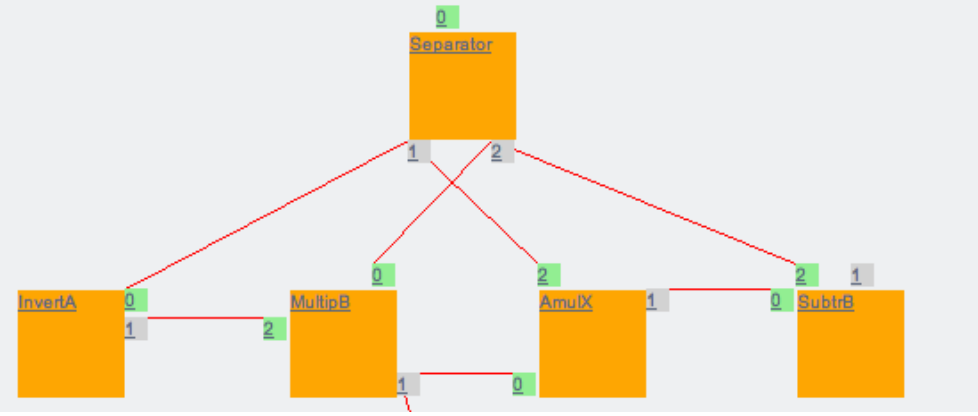

Figure 1 Directed Acyclic Graph Example

Figure 1 shows an example of a DAG. The orange rectangles are jobs, and the grey squares are output ports and the green squares are input ports. The edges are files that are supplied by the output ports to all connected input ports. This structure allows the workflow to be executed in a parallel manner by scheduling jobs for execution as soon as their inputs become available, and executing the job as soon as there is a resource available for it. Multiple jobs from the same workflow can therefore be executed in parallel[9].

In combination with repository technologies a configured workflow can be executed an arbitrary number of times, each execution of which is a workflow instance. In a similar manner, jobs that appear in the DAG, to be referred to as abstract jobs, can be executed multiple times, across multiple workflow instances, or a single abstract job can be executed many times within the same workflow instance, when using special ports [13]. Those ports cause the job to be executed for each of some combination of the inputs.

1.3: Portals

A portal is a web system that provides an interface for accessing services such as a grid portal or a gateway platform. Originally all major portals started out as Grid Portals and were later extended to support other infrastructures, such as desktop Grids. The portals act as portlet containers and provide basic functionality to incorporate a portlet framework.

The WS-PGRADE Grid Portal is the second generation of the original P-Grade portal. The portal allows creation and submission of workflows on multiple DCI's. The portal uses

the Grid User Support Enviornment (gUSE) to provide the gird functionality. One of the services is the gUSE repository which stores the workflow objects to be downloaded and further developed. Furthermore it provides a forum for collaboration and enables workflows to be shared across the community [10].

1.3.1: WS-PGRADE Grid Portal and gUSE

The WS-PGRADE Grid Portal and Grid User Support Enviornment(gUSE) are both products developed by MTA-SZTAKI LPDS branch. The WS-PGRADE Grid Portal is the second generation of the P-Grade Portal. It is a "web based environment which provides tools for the development and execution of workflow based grid applications." WS-PGRADE added capability to better handle both parameter study and workflows and the internal structure changed to be a modular, service oriented architecture based system. This change was implemented through the development of gUSE. gUSE provides a graphical environment that a user can define and execute grid applications on, using the WS-PGRADE as a user interface [9].

1.3.2: Liferay

Liferay Portal was created in 2004. It is a software platform for building websites and web applications [4]. It can be used for web, integration, collaboration and social application platforms. Liferay is developed by a large open source community as well as professional interactions. This makes it both flexible and innovative.

The Liferay portal is used in the WS-PGRADE Grid Portal as part of the user interface framework. As WS-PGRADE uses the Liferay framework, our user interface was built as portlets that can be viewed on Liferay.

1.3.3: Portlets

"A portlet is a Java technology based web component that processes requests and generates dynamic content." Portlets are used as plug-ins to an existing user interface to provide different features. This allows a website to be customized for each type of user as well as provide different content. A portlet is managed by a request and response paradigm, and normally is intractable through its forms and links.

A portlet is managed by the portlet container, Liferay in this project, which provides them with the runtime environment. It contains and manages the lifecycle as well as storage and preferences. The container and portlet can be separate entities or built together. [1].

1.4: Metrics

Metrics are a measurement of performance, efficiency or other statistics in an application. For the WS-PGRADE Grid Portal there are numerous metrics for the different aspects of the system. We defined metrics that deal primarily with usage statistics.

Among the metrics we were able to calculate are:

- Average job completion time
- Average time jobs are in different states
- The standard deviation for the times
- The number of jobs
- Number of workflows
- Running failure rate
- Number of failed jobs

1.5: MTA SZTAKI

MTA SZTAKI is Hungary's largest and most successful information technology research Institute. The name is an acronym for "The Computer and Automation Research Institute, Hungarian Academy of Sciences" in Hungarian. It is governed by the Hungarian Academy of Sciences and is supervised by the Board of the Institute [11]. It was founded in 1964 and has more than 300 full time employees.

The main task for the institute is to "perform basic and application-oriented research in an interdisciplinary setting in the fields of computer science, engineering, information technology, intelligent systems, process control, wide-area networking and multimedia." [11]. Also they do contract-based research, development and training as well as provide support for domestic and foreign industrial, governmental and other groups. They are active in both graduate and undergraduate education offering lectures and classes as well as providing opportunities for students to participate in the work at the institute.

The institute is a part of the European Research Consortium of Informatics and Mathematics and a member of the World Wide Web Consortium. They have worked on projects for both Hungarian companies, such as Paks, a Hungarian Nuclear Power Station, and international companies such as General Electric, the National Aeronautic and Space Administration, and the Office of Naval Research One of their main research areas is cluster and grid computing.

1.5.1: LPDS

The Laboratory of Parallel and Distributed Systems (LPDS) is a branch of MTA SZTAKI that specializes in grid technologies. LPDS is a member of the Hungarian Grid Competence Center and the National Grid Initiative. The department is headed by Proffessor Dr. Péter Kacsuk, a renowned expert in the field of Grid computing and co-editor-in-chief of the Journal of Grid Computing [12]. LPDS has produced five projects, the most prominent being the WS-PGRADE Grid Portal.

LPDS participated in the CoreGRID Network of Excellence and works as a project member in all the phases of the largest European grid infrastructure project (EGE, EGI-Inspire). Furthermore they helped establish the Hungarian Virtual Organization of the European Grid Infrastructure (HUNGRID) extended with the WS-PGRADE Grid Portal. They are also involved in many more projects as well both nationally and internationally.

They have two goals in grid research:

"To provide efficient software development tools and high-level services together with customizable scientific gateways based on workflows (P-GRADE Grid Portal, gUSE) for harvesting the most wide-spread grid infrastructures based on gLite, Globus, and BOINC"

"To offer easy-to-maintain middleware solutions (SZTAKI Desktop Grid) and technologies for interoperability (3G Bridge) that enables cost-efficient alternative platforms for scientific and business applications."[10]

2: Methodology

In order to determine the requirements for our system we progressed through a series of steps to determine what metrics we wanted to make available to the user, what data we had to store in order to provide those metrics, and how we had to transform the data we received into the data we needed to store. Furthermore we explored different methods of displaying these metrics to the user.

2.1: Architecture

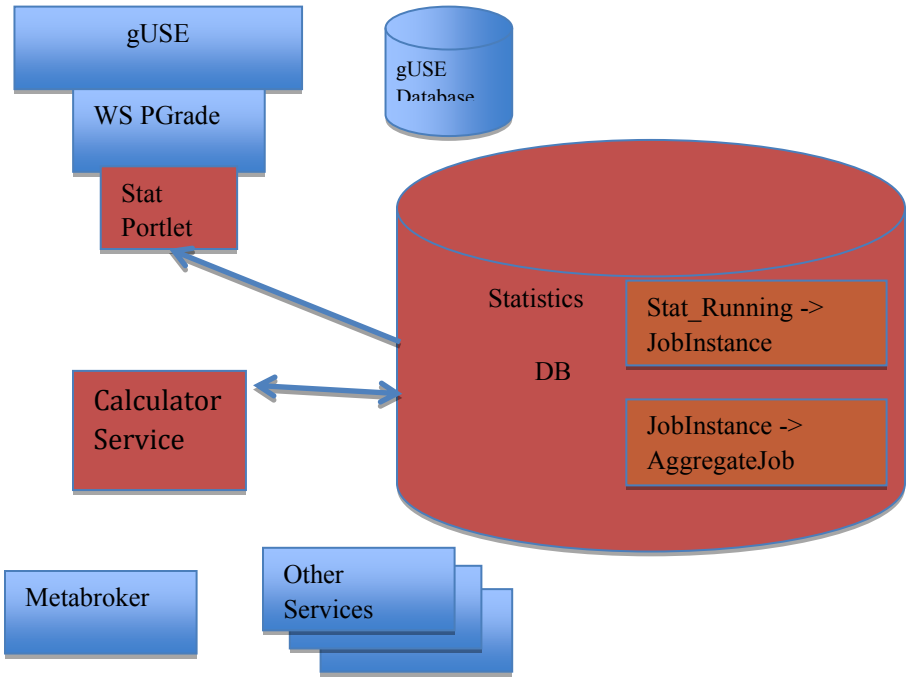

Figure 2 System Architecture

Figure 2 reflects the architecture for the system, with our proposed components in red and orange. The proposed components have to receive job status data from gUSE and group it in an efficient and meaningful manner. To do this, the statistics database will handle grouping of data on the job level, and the proposed calculator service would use the grouped data to calculate statistics and store the calculated values in another database structure for the calculated statistics. The calculated statistics tables would be read by the portlet in order to be displayed to the user.

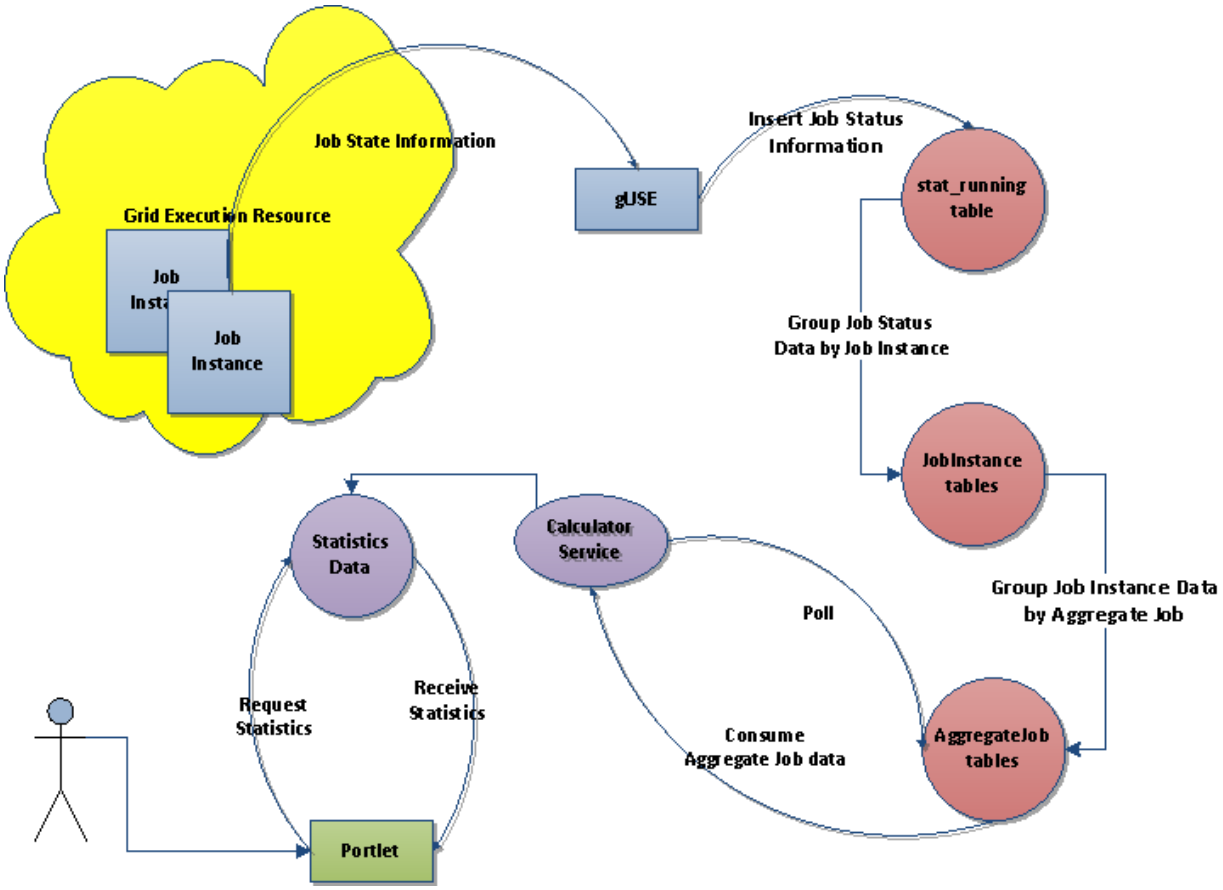

Figure 3 Data Flow Diagram

The above figure shows how data flows through the proposed system. The information starts in the statistics database as entries in the `stat_running` table, which is populated by a gUSE service. The stat_running entreis describe the current state of the job at a specific point in time. These values are then combined using MySQL database triggers into structures based off of job instances run on the grid. The job instance values then are grouped again into a structure called aggregate jobs which are a combination of several job instances that share the same job name, workflow instance and resource. There also exists a web service,the calculator service, that consumes the aggregate jobs, and calculates the metrics for the user. The calculated values are then available to the portlet for display to the user.

Overall, this design allows our services and database to be completely isolated from the gUSE systems that allows the performance to be controlled independently. The

exception to this would be the constructs created for the portlet to provide useful menus to the user.

2.2: User Interface Requirements

The UI requirements included functionality requirements and usability requirements. The functionality requirements included being able to show the metrics gathered, accessing the database, having similarity to the rest of the portal, and creating a way to navigate the data. Showing the metrics gathered required providing a layout and a table structure as well as offering graphical representations of some statistics. The metrics the user needed to be able to view were on several layers listed below. Each layer had to have the same layout for organization purposes as well as function in the same manner, even though the data accessed was different. Accessing the database required a way to retrieve the data. To maintain similarity with the rest of the portal it was necessary to study the previously completed sections. Finally to navigate the data required setting up choice lists as well as menu buttons. The menu buttons were main navigation, reaching all the top levels of metrics such as portal, user, DCI, and concrete workflow. The choice lists required populating the list with what was available. Furthermore it required that the user makes choices either with a drop down menu or a user filled input box.

The usability requirements included general user interface standards such as size of text or coloration. Other standards include arrangement, readability, comprehensibility, and usability.

1. Users may view metrics about:
    - The WS-PGRADE Grid Portal
    - User
    - DCI
    - Resource
    - Concrete Workflow
    - Workflow Instance
    - Abstract job
2. Users may choose the navigational buttons:
    - DCI
    - User

- Concrete Workflow

3. Users may select individual:
    - DCI's
    - Resources
    - Concrete Workflows
    - Workflow Instances
    - Abstract Jobs
4. Users may compare multiple Concrete Workflows

2.2.1: Use Cases

For the interface there were multiple levels of metrics. Administrators, as mentioned before, are interested in the overall portal statistics as well as the DCI and resource levels. While our system provides the data to all users, the differences between them will mean some levels of data will be more useful to a particular type of user. For example an administrator may be interested in the amount of jobs run on a certain resource; while a user may be more interested in the amount of time there workflow took. For this reason the data was divided into the multiple levels.

The user can view the levels by choosing different menu options. The portal and user levels assume the statistics to be displayed were the current portal and user; the other levels require a choice of what object to be displayed. This is because there are multiple options, for example a user can have many concrete workflows, and it is not possible to easily display all of them.

The multiple levels and choices allows both administrators and users to view only the statistics they wish to see, without having to deal with an overload of information. Overall this design works for the system because there is no need for a user to see more. The user may only view their statistics because the other levels of statistics provide for comparison. The other choices provide a way to view statistics on individual objects instead of receiving an overload of information. A user can compare DCI's to select which one has been performing the best in the past and choose individual resources if they wish to view another level in. The same works for concrete workflows. The user can choose one and then expand upon it by selecting an abstract job or workflow instance.

2.2.2: Use Case Diagram

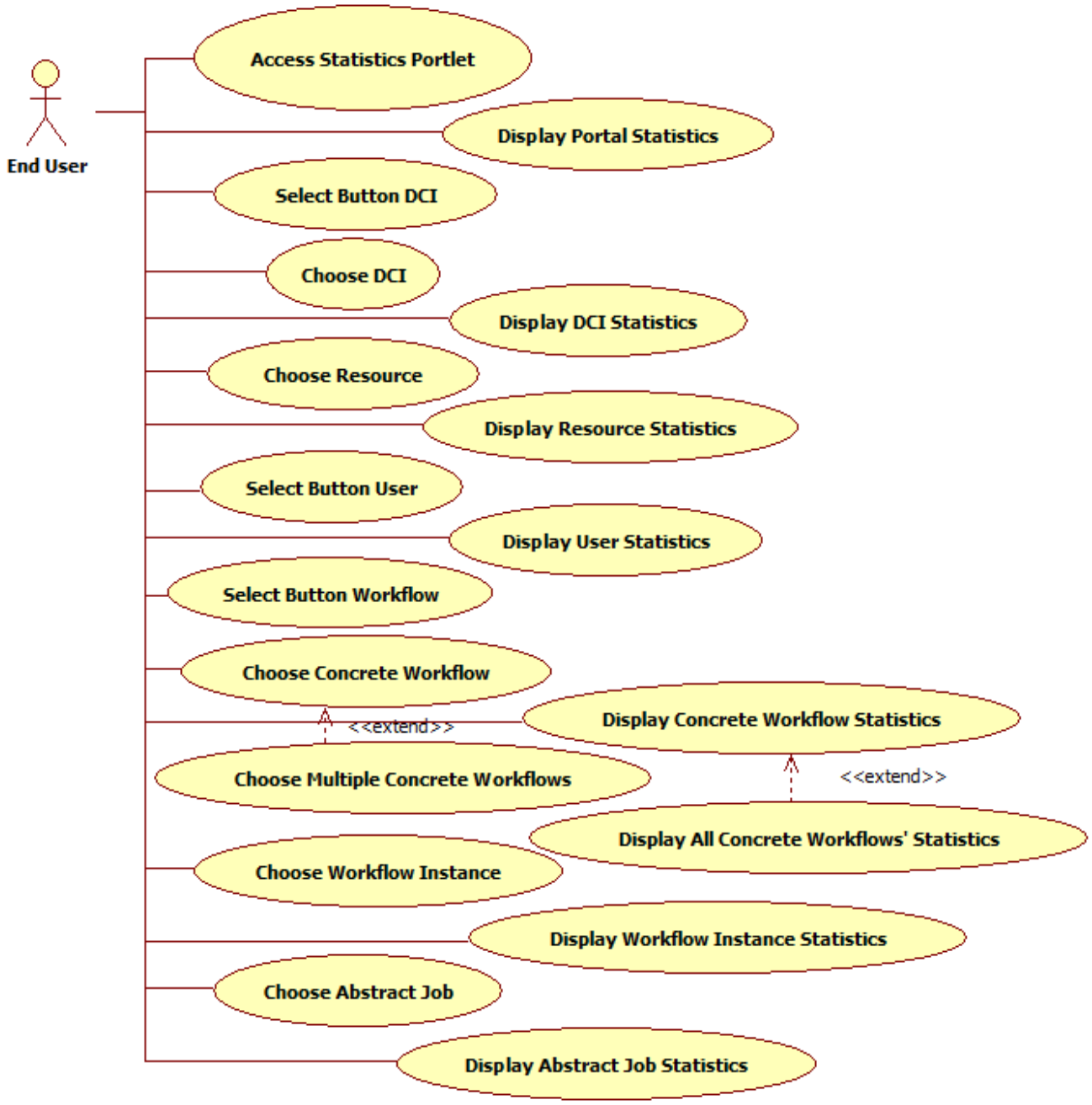

Figure 4 Use Case Diagram

For this system there is only one actor, an "End User". This represents anyone using the system such as an administrator or normal user. Each user can perform the same

actions regarding navigation and viewing statistics. The diagram below shows what is possible.

2.2.3: Sequence Diagram

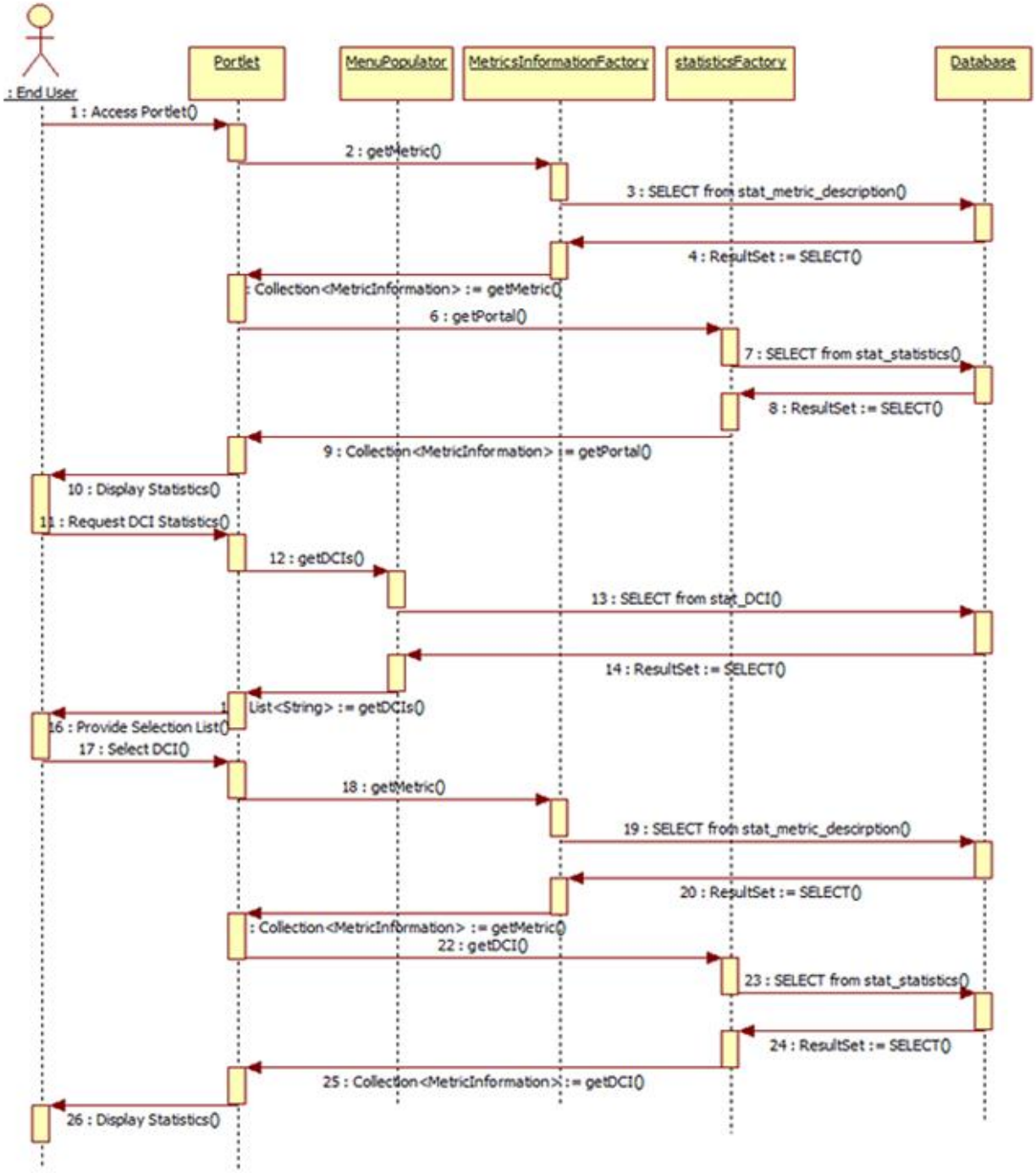

Figure 5 Sequence Diagram DCI Statistics

Figure 5 is a sequence diagram that demonstrates one path to get statistics, in this case for DCI metrics. This path is similar for all the levels. The portal statistics are displayed first and then the user needs to make a choice what to access next. The portlet serves as the user interface for the end-user and provides the options for the user. The Menu Populator is responsible for providing a choice list for the user in applicable cases. Metrics Information Factory provides metric descriptions such as the name and units of the possible metrics. Statistics Factory retrieves the data for the given metric description and the database provides the data for all the objects.

The end user, either an administrator or user, accesses the portlet, which accesses `MetricsInformationFactory`. The factory queries the database table `stat_metric_description` which sends the results back to the factory. This step returns a collection of the metric information back to the portlet. Next the portlet sends the information to the `StatisticsFactory`. This factory queries the database for portal metrics and receives the result set, which populates the collection of metric information with data. The information is then sent back to the portlet. The data is than displayed to the user.

At this point the user can request to view DCI statistics. The portlet accesses the object `MenuPoulator`. `MenuPopulator` accesses the database to receive a list of possible DCI's and returns it to the portlet. The portlet produces a selection list for the end-user. Once the user makes a selection, the path is the same as with portal metrics, except with DCI information.

2.2.4: User Interface Candidates

Before starting on the programming aspect of the user interface we created multiple candidate designs to present as potential candidates for a user interface. The designs were based on the assumption that there would be only one page to display all the data. Furthermore, they were designed before we knew the amount of data we could retrieve and before we had directly interacted with the system. The two designs below are the closest to the final design.

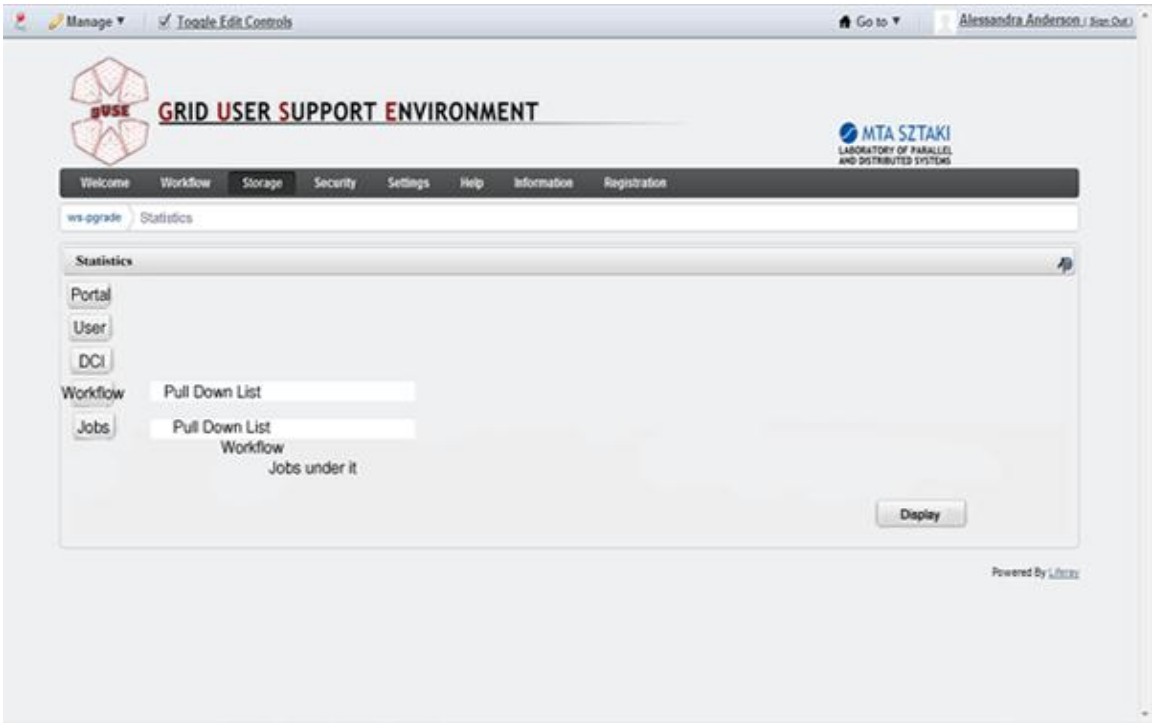

Figure 6 Candidate Design 1

Design 1 has six buttons the user could select for the level and then hit display button to get to the statistics. Whenever a user would choose a button it would appear to be pushed in to indicate it was selected. The button remained depressed until the user either deselected it or hit display. To select a job or a workflow the user would be offered a drop down list to choose from. Again they could select *multiple* to display at once by highlighting more than one.

Advantages	**Disadvantages**
Simple for the User	2-Step Process to see statistics
Clean and uncluttered	Looks unfinished

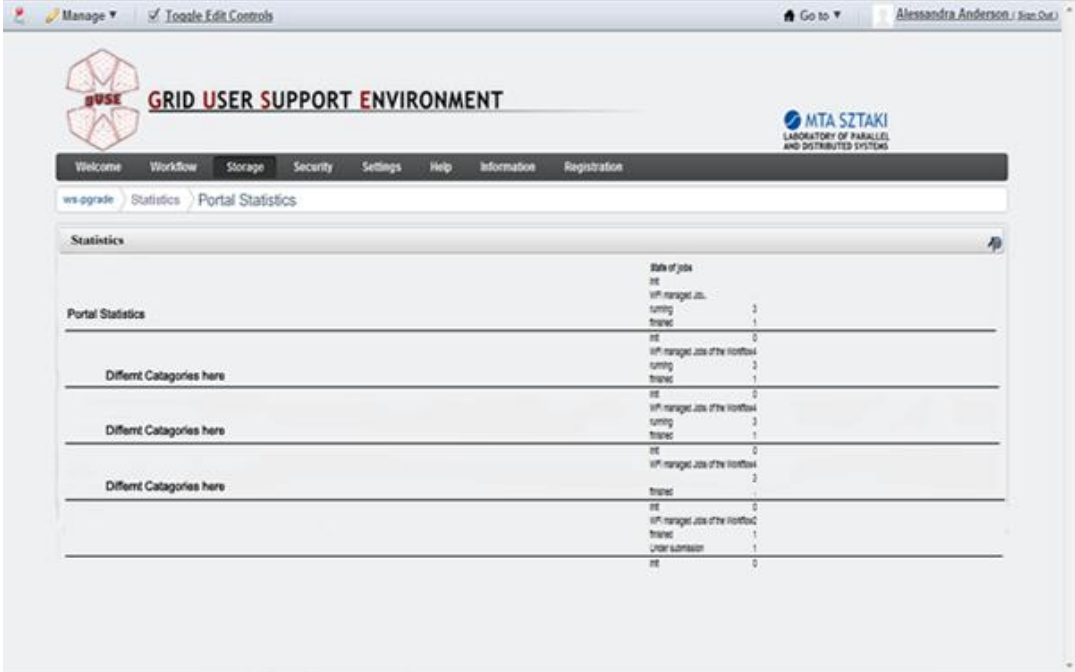

Figure 7 UI Candidate Display Design

Figure 7 shows a candidate display design. This design displays each metric in categories and sub-categories. A sub-category is a grouping of statistics for example times. For each choice the user had selected a main category, such as "Portal" would be generated, and sub-categories of each type of metric would be created below.

Advantages	**Disadvantages**
Metrics Available Right away	Potentially a lot of scrolling
Clean	Takes up a lot of room
	No Customizability

The final design was loosely based on the two above. These designs evolved into the final design as we progressed through the project. The principle of selecting workflows and abstract jobs as well as separate categories for displaying was still incorporated into the final product. Furthermore they were useful for discussions on how the final interface should look.

2.2.5: Final Design

After reviewing the original designs, the final design was proposed. This design consisted of creating multiple pages to display each level of statistic on its own page. The pages were divided into the different levels: a page for portal, user, DCI, and concrete workflow. The user could choose up to three concrete workflows to display at once. For DCI the user could choose to view individual resources on the selected DCI and for concrete workflow the user can choose either abstract job or workflow instance metrics to view. After the portal is accessed the portal metrics are displayed automatically. ,

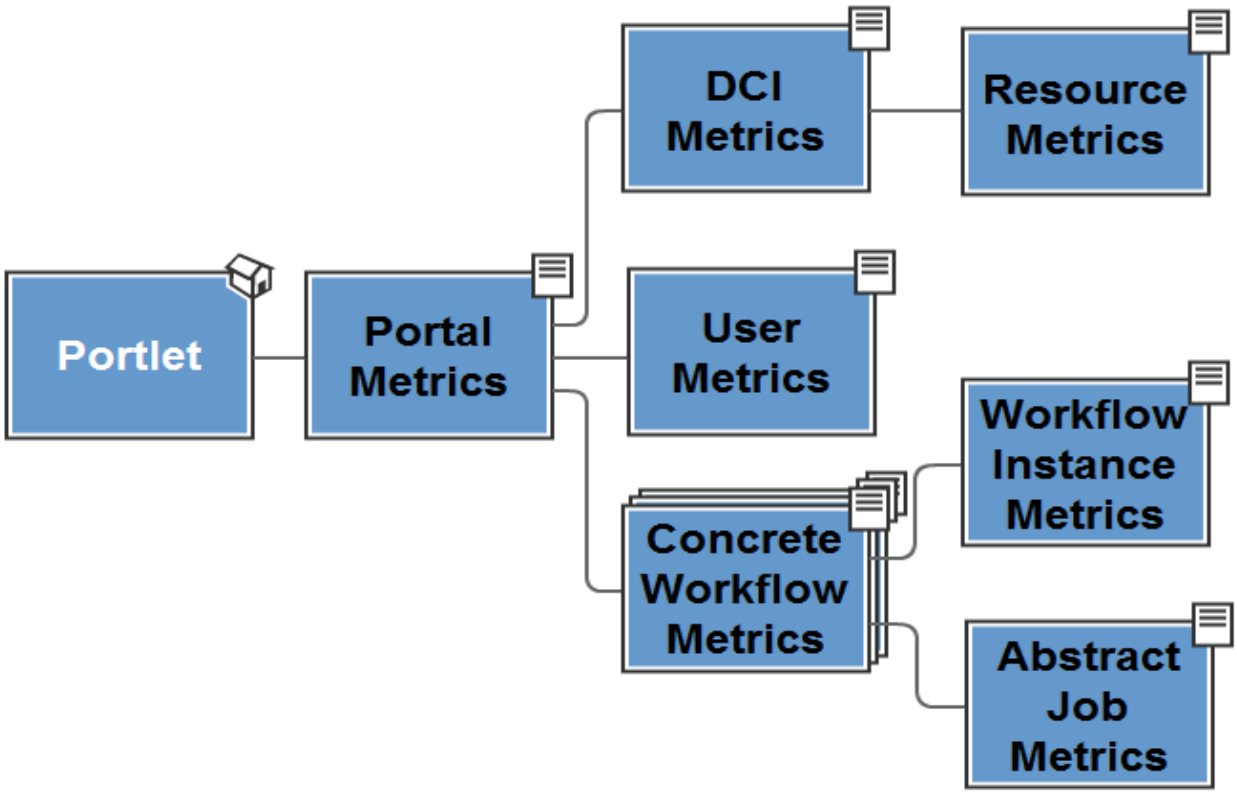

Figure 8 Site Map

Figure 8 shows the final site map. The user accesses the portlet which shows them user statistics. From there they can navigate to DCI, user, or concrete workflow metrics. From there the user can enter one more layer, viewing resource, workflow instance or abstract job metrics. Back End Requirements

Another area of functionality for this project was in maintaining the database structures that support the system. The data had to be aggregated in such a manner that we did not consume all of the resources of the database. However, there was a drawback to aggregating data as detail was lost with every aggregation operation. Therefore, in order to provide as much useful data as possible, the data was organized into aggregate jobs units, which combined the data for each abstract job for each workflow instance into one structure. This allowed us to aggregate all jobs involved in a parameter study into few entries as they are all similar. Furthermore, in order to provide data to compare grid resources we also divided aggregate jobs on the resource that it was executed on.

In order to remove the load of statistic calculation from the grid portal, we also need a method of pre-calculating those statistics that would be required of us. This service must use the aggregate job entries and use them to calculate the metrics. With this in mind, here are the requirements for the data maintenance portion of the project.

1. The system shall group job instance data.
2. The system shall group job instance data with the same job name, workflow instance and computing resource into constructs called Aggregate Jobs.
3. The system shall pre-calculate statistics from aggregate jobs for the user interface.

2.3: Data Aggregation

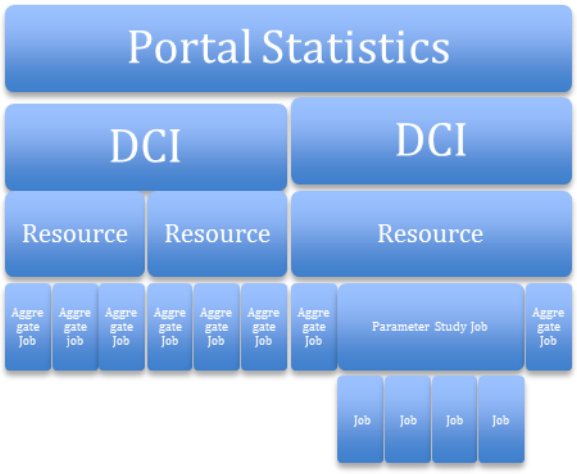

Figure 9 Data Composition Diagram

Figure 9 shows a high level example of the method of aggregating our data into statistics. For each layer, statistics can be generated through some combination of the layer below, until the “Aggregate Job” layer. In the diagram, the only data that is being stored is the data for an “Aggregate Job” which is one of two things. If the aggregate job refers to a parameter study node, such as the case of “Parameter Study Job”, in the diagram, aggregate job stores statistics about the aggregate of all of the jobs that compose it. Otherwise, there is a one job instance to one aggregate job relationship. This allows us to significantly reduce the volume of data stored.

The aggregate job structure can therefore be used to generate statistics about larger constructs. For instance, Figure 17 shows how statistics about a resource are composed by aggregating statistics about all the aggregate jobs that have been run on that resource. Furthermore, DCI (Distributed Computing Infrastructures) statistics can be aggregated from all the resources that compose it. There are similar paths to aggregate statistics about users, workflow instances, abstract jobs, and concrete workflows.

2.4: Design Concerns

During the design, one of the main issues that was presented the amount of memory use and CPU load on the gUSE and WS-PGRADE Grid Portal servers. Our goal was to keep any load on these servers to a minimum so that the portal operation would not be impacted significantly. This was one of the primary reasons for our calculator service to be a separate web service from gUSE. We also designed our database components to function on a separate database from the gUSE database if called for.

The main concerns for the front end was how to display the amount of data provided in a simple and meaningful way that did not require too much hardcoding. Furthermore we wanted to have a simple way to change what was displayed without having to touch the code. Finally we wanted to be able to display some of the data graphically.

3: Implementation

For this stage of the project we sought to implement our system and add a portlet to the user interface to the WS-PGRADE Grid Portal. We first implemented the changes to the database which included defining schema changes and stored procedures. Having the database defined allowed us to concurrently implement the user interface and the calculator service.

3.1: Database

The database component of the system focused on creating and modifying database structures in order to aggregate the data from the gUSE system. Our main concern was the scale of the data that we received, which consisted of many entries for each job instance run on the grid. As the number of jobs that are run could be very large due to the nature of parameter study workflows we determined we must consume these entries upon their entry into the database. This was accomplished using database triggers, which execute a routine in conjunction with SQL `INSERT` or `UPDATE` statements.

There were three table structures maintained by the database.

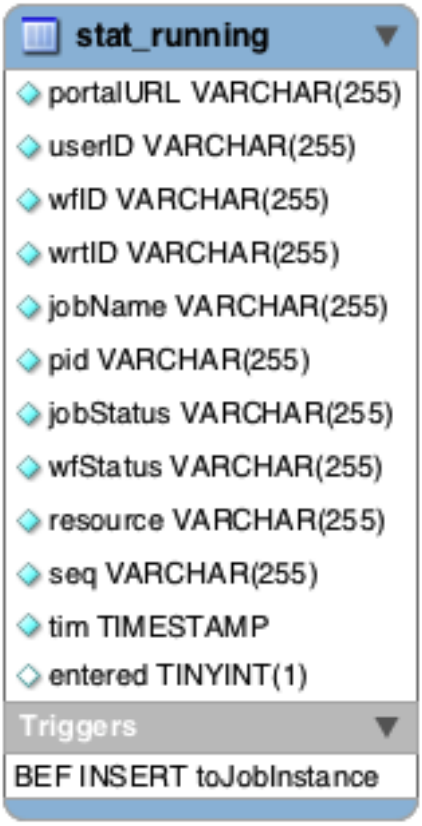

Figure 10 stat_running table description

First was the `stat_running` table, which received data from gUSE in a polling manner. For each job being run on the portal, the portal would periodically query the job's status and record the information in this table. Therefore, this table has many entries for each job executed.

The `stat_running` table was consumed using database triggers that executed whenever a row was inserted into it. That trigger would create or add to data to the next intermediate table structure which grouped data by job instance.

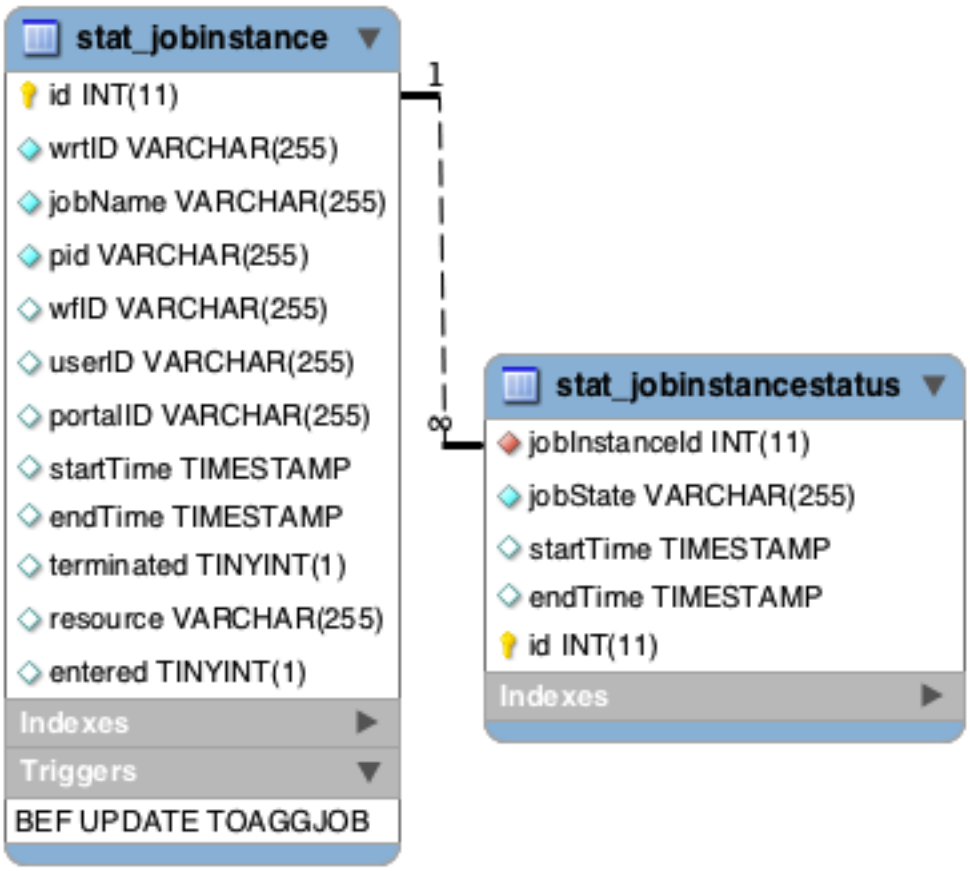

Figure 11 stat_JobInstance and stat_JobInstanceStatus

The `stat_JobInstance` table structure maintained data for each of the states that the job touched. There are currently 23 possible states. With the shown structure it is only required to maintain information about the states that are used. Our system however is built with the assumption that the number of states can change. Also, one of the states was added for our system. This structure also handles the case of the loops in the state diagram for jobs, by allowing multiple entries for all of the states.

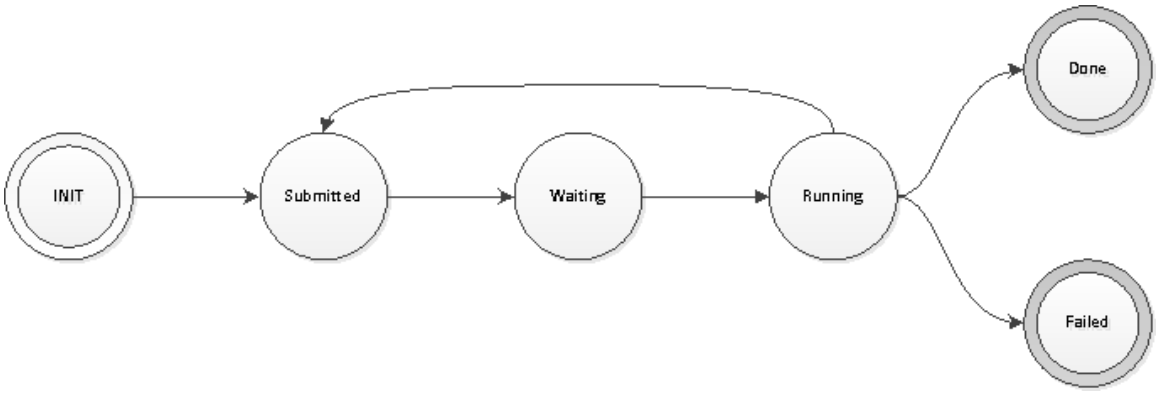

Figure 12 Simplified Job State Diagram

Figure 12 represents a basic subset of the graph of states that jobs may traverse during their execution. The full list of possible job states is available in the appendix. Primarily we store data on the transitions between states, and combining different states allows us to draw conclusions about where in the system the job is waiting.

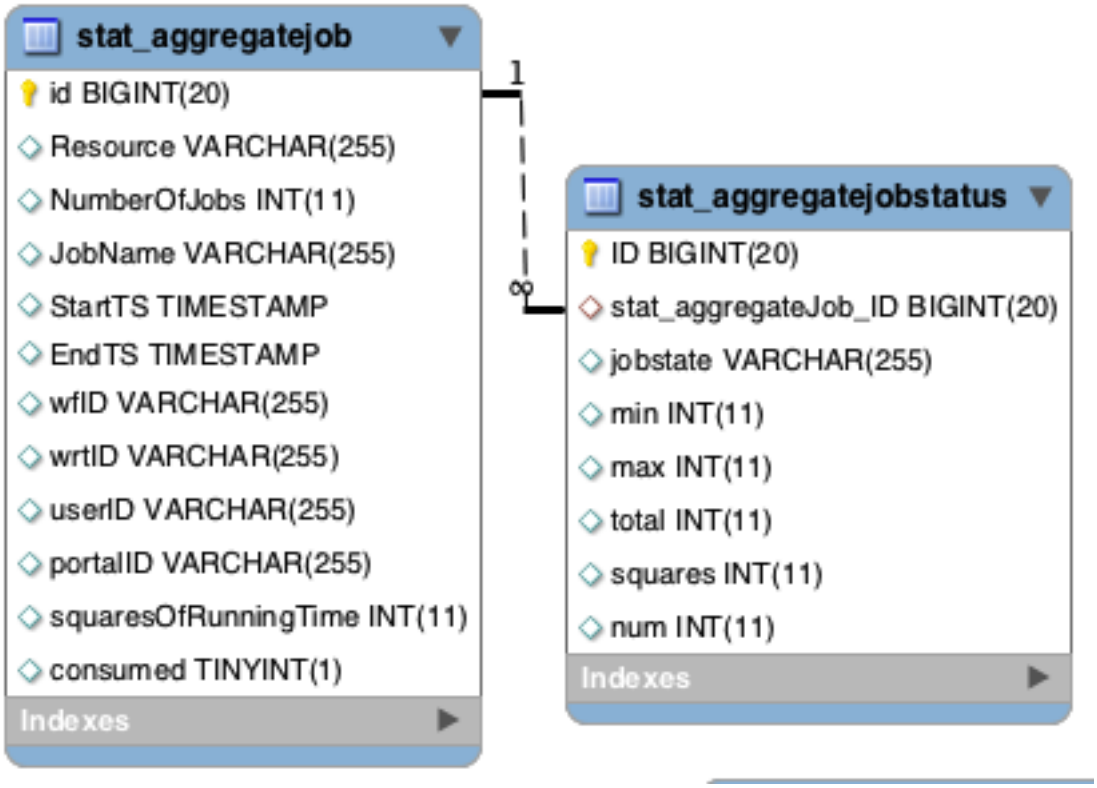

Figure 13 stat_AggregateJob and stat_AggregateJobStatus

Figure 13 shows the final step in the database component, the aggregate job structure. Aggregate job combines data from several job instances that share some data and combines the jobs into one structure. All of the job instances combined into an aggregate job share the same job name, workflow instance, and execution resource. The same job name and workflow instance means that the job instances share the same executable routine. Enforcing the same resource allows comparisons between different grid resources executing the same job.

The data stored in the aggregate job tables is similar to the data from the job instance table. The main difference is that it is structured to combine several job instances. For each state that any of the job instances visit data required to calculate the average time, the number of entries into that state, and the standard deviation are stored. One of the requirements for this table was that the only metric information that we store would be calculable with only the previous metric value and information about the values to add to it. Storing data that satisfies this requirement allows us to calculate aggregate job data incrementally, adding one job instance at a time.

3.2: Calculator Service

The calculator service's goal was to retrieve the data from the aggregate job tables and calculate the relevant statistics for each of the seven levels we are providing: portal statistics, DCI, resource, user, concrete workflow, workflow instance and abstract job. The service did this in three steps, first it queried a set of aggregate job entries, then it calculated the changes in the statistics for each row for each of the seven levels of statistics that needs updating. Finally, it then performs an update on the statistic database tables. The calculator service also managed some database clean up for the database component.

For the querying of the aggregate job entries there were several concerns. As our calculator was being implemented as a simple web service we wanted it to only pull in a manageable subset of the aggregate job entries, as the design called for the subset to be stored in memory. This was addressed through a `LIMIT` clause on the SQL query. Another concern with the query was a race condition with the database component. As the database component needs to write to the `stat_AggregateJob` table whilst the calculator needs to read from it, we had to implement a guard that would allow the calculator to know when a `stat_AggregateJob` entry is complete, or that it will not have any more job instances

added to it. This was solved by only querying aggregate jobs that the workflow instance that executed them is terminated.

The calculation step had to consume the aggregate jobs and calculate what effect they had on the pertinent statistics. For each aggregate job the change in statistics is calculated for each portal, user, DCI , resource, concrete workflow, workflow instance, and abstract job , using the identifier shown in the table below.

Table 1 Statistic Level Identifiers

Statistics Level	**Identifier**
Portal	Portal URL
Resource	Resource URL
Concrete Workflow	Workflow ID (wfID)
Workflow Instance	Workflow Instance ID (wrtID)
Abstract Job	Job Name and Workflow ID (jobName and wfID)
User	User ID

The change in the statistics are then stored in the database using some combination of SQL updates and SQL inserts for values that do not exist yet.

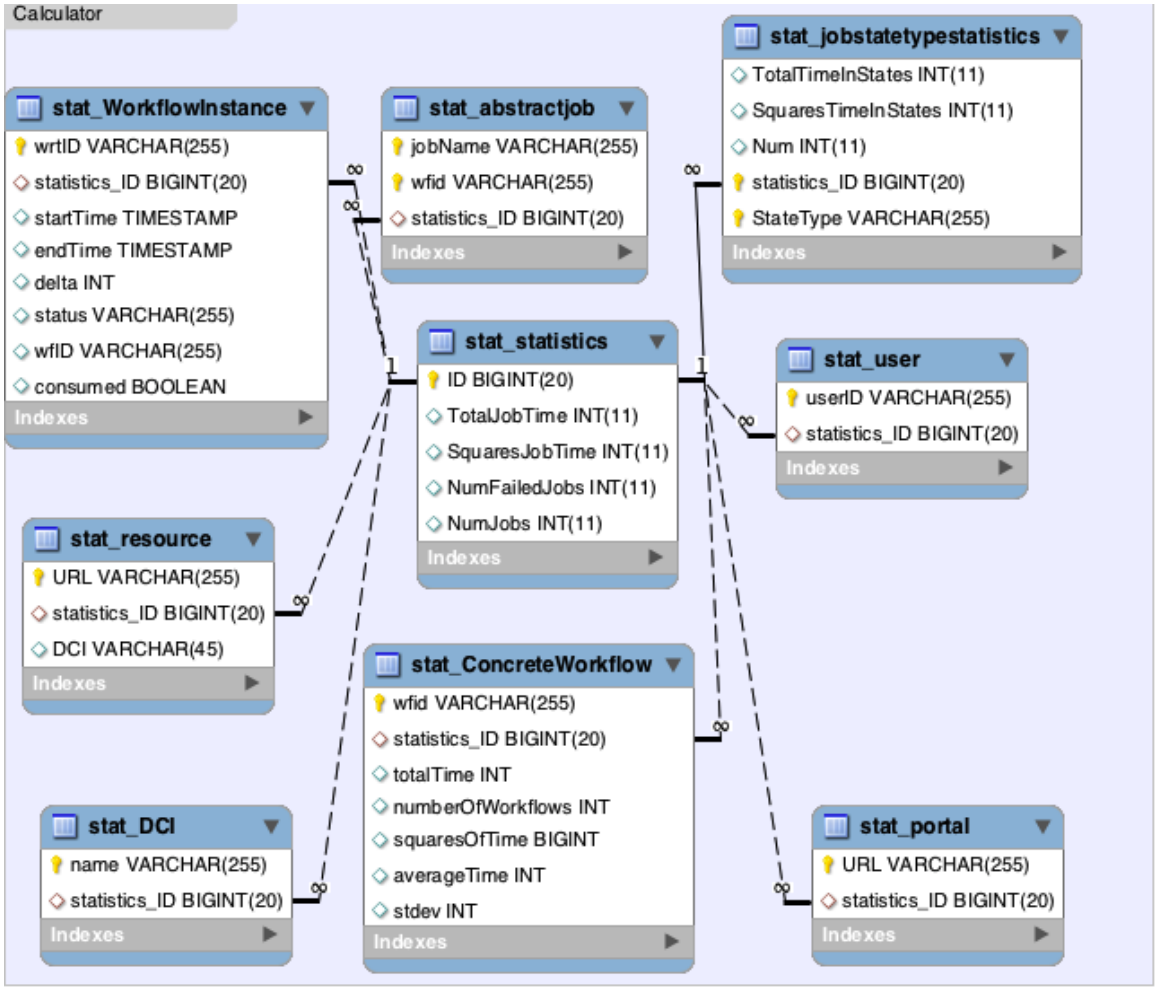

Figure 14 Calculator Database Structure

The above table structure holds the final statistics for our system. This structure allows us to isolate the storage of the statistic values, such as average, from the identity, such as resource URL. This simplified the table structure by removing common, shared columns into a separate table. The exception to this is workflow instance where we store the start and end time for the workflow instance, and concrete workflow where we store statistics about the workflow as a whole. However, the difference in workflow instance is not maintained by the calculator and instead is maintained by the database component. While this is not the ideal place for the responsibility, it was necessary because we use that information to know when a workflow is complete so that the calculator only pulls aggregate jobs from complete workflow instances.

The final task of the calculator service was the database cleanup. Due to the triggers handling the data aggregation in the database component, it was impossible to delete consumed entries from the `stat_running` and `stat_JobInstance` tables. Instead, we were only able to flag the offending rows for deletion. Therefore, since the calculator is already polling that database, it also runs a SQL delete query to remove the unneeded entries.

3.4: UI

The front end of our system was the portlet integrated into the WS-PGRADE Grid Portal. To accomplish this we used a multitude of tools and Liferay. Liferay was used as part of our development environment to upload the portlet and test its interactions. The creation of the portlet was done in multiple iterations eventually ending with the final product.

3.4.1: Tools/Languages

The user interface was an additional portlet added to the preexisting webpage. For this four languages and tools were used: HyperText Markup Language(HTML), JavaServer Pages(JSP), JavaServer Pages Standard Tag Library(JSTL), Java Script and Google Chart Tools.

1. HTML is the predominant language for the design and display of webpages. It is used to create structure, formatting, and functionality in a webpage.
2. JSP is a technology that enables the design dynamic Web pages and separates the user interface from the content generation which allows a Web designer to change the page layout without altering the underlying content[7].
3. JSTL: “A collection of tag libraries that implement general-purpose functionality common to many Web applications.” [7]
4. Java Script is an object-oriented scripting language that is used for web development to create more interactive webpages.
5. Google Chart Tools or Google Chart API is a tool that allows the creation of charts from data and embeds it in a webpage. The embedded data must follow the formatting parameters in an HTTP request, and Google than

returns a PNG image of the chart[14]. We used this tool because it allowed simple creation of dynamic graphs.

3.4.2: Implementation Process

The UI implementation was done between three main stages with multiple iterations within them.

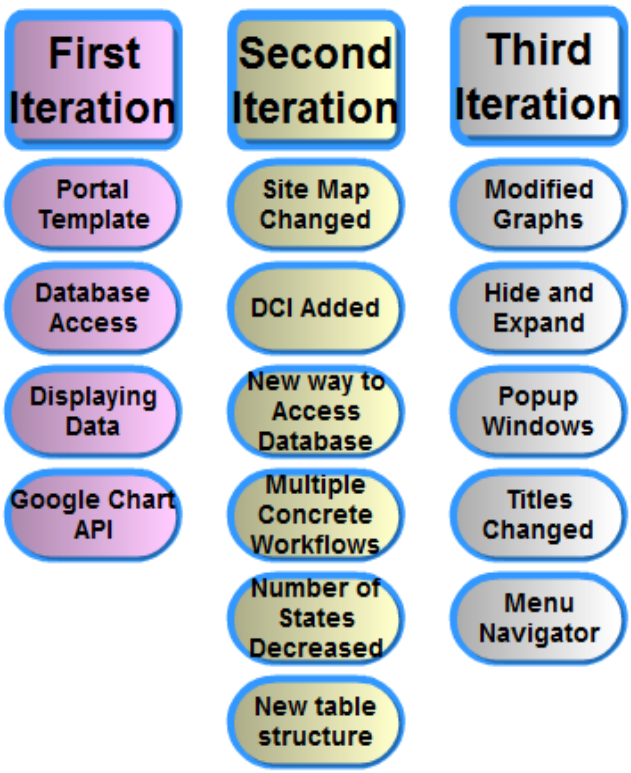

Figure 15 UI Implementation Graph

The figure above highlights each iteration and the milestones within it.

3.4.3: Iterations

The UI was developed in three iterations, with numerous milestones for each one. The first iteration consisted of creating a template for the portlet that could access Liferay, accessing the database, displaying the data and adding the Google Chart API.

Figure 16 Original User Interface

Figure 16 shows the original design, seen below, displayed metrics in rows.

The second iteration incorporated many changes. First, the site layout was changed to the final version. Second, DCI was added to the levels of metrics. Third, a new way to access the database was implemented. Fourth, the ability to select multiple concrete workflows was added. Fifth, the number of states was decreased to five instead of seven. Finally, a new table structure was added. The new layout is seen below.

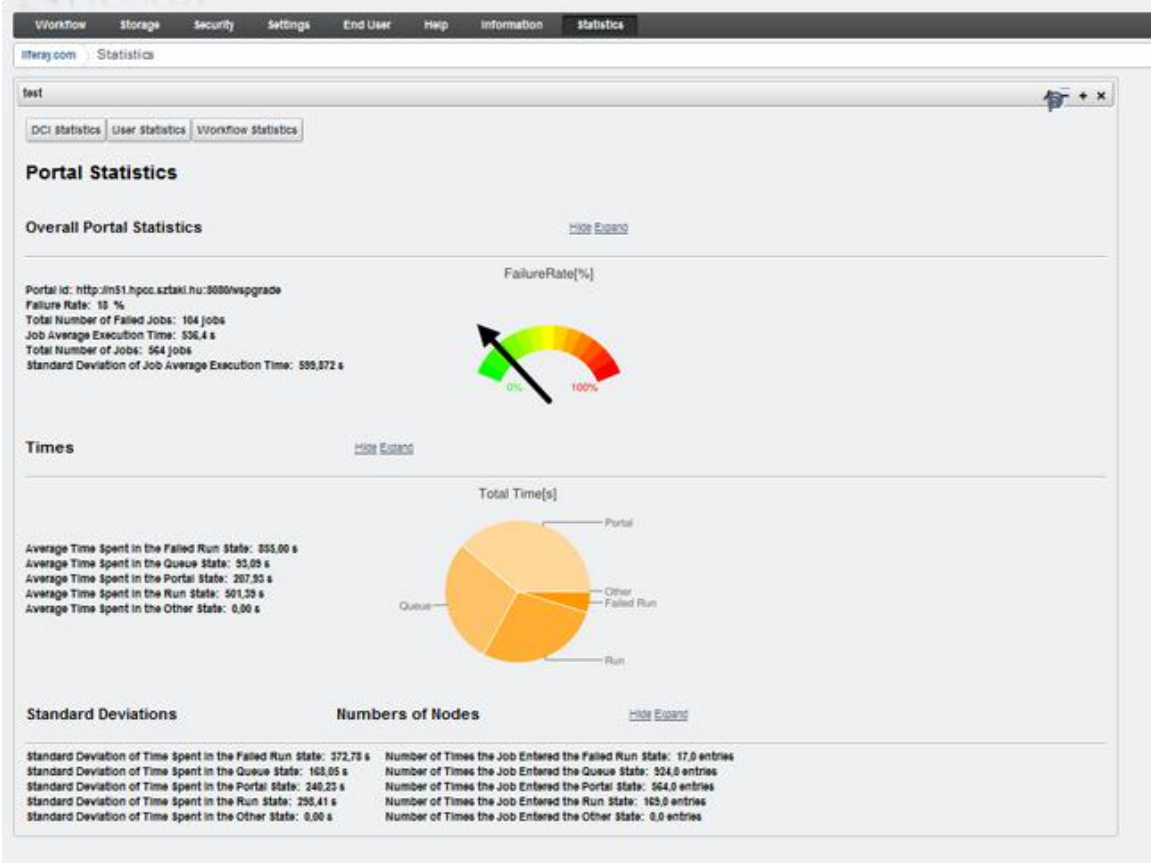

Figure 17 Second Iteration User Interface

The table that was added to the database, `stat_metric_description`, created a simpler way of presenting the data. This table was comprised of nine columns: `column_name`, `pretty_name`, `category`, `units`, `precision`, `source`, `for_level`, `statetype`, and `id`. The `column_name` referenced what column the data was being accessed from the source table. The `pretty_name` and `units` columns were the description and units respectively that would be shown on the portlet. The `precision` column was the number of decimal places that would be shown. The `id` was both the primary key for the table and was also used for ordering of statistics within a category. `for_level` specified what the statistic was good for, as some metrics only worked for certain levels and the `statetype` column allowed us to set one of the five state types we were using. Lastly the `category` column allowed statistics to be grouped together so they could all be displayed with a single call. The category dictated which metrics would be displayed together, for example times in state types was one category. By extracting the

presentation information from the database this extra table cut down the amount of hardcoding considerably and made the system overall easier to modify.

The final iteration incorporated small changes to achieve the final product. Frist, both graphs were modified to better display the data. Second, hide and expand options were added to each category of statistic. Third, both abstract job and workflow instance were changed to be displayed in popup windows, instead of on a separate page. Finally, a menu navigator was added at the top of each page, and the titles for categories changed to better describe the metrics.

3.4.4: Final Product

The final product showed portal metrics when the user accessed the portlet. Users could choose, from a top menu, to view DCI, user, or concrete workflow statistics. From DCI and concrete workflow the user could enter another level and view resources, workflow instances, and abstract jobs. Each level was displayed in the same format except for workflow instance and abstract jobs which were displayed in popup windows.

The final metrics that were shown fell into four categories: overall statistics, runtimes in states, standard deviation, and number of times a job was in a run state. The first category contained metrics such as the name, overall time, and failure rate. Following categories were dependent on the states a job could enter. The states were run, failed run, queue, portal, and other. These states were the combined states of all of the job states available in the system. We created the pooled states because the user would not be interested in all the states available. The run state was when a job would successfully pass through to completion. The failed run state was how long the job would loop through the states, as it was possible to go from run back to queue, or another state. The queue state was how long the job was waiting at a resource and the portal state was the time spent on the portal before being submitted to a resource. Finally, the other state encompasses any states that are not covered in the other joint states.

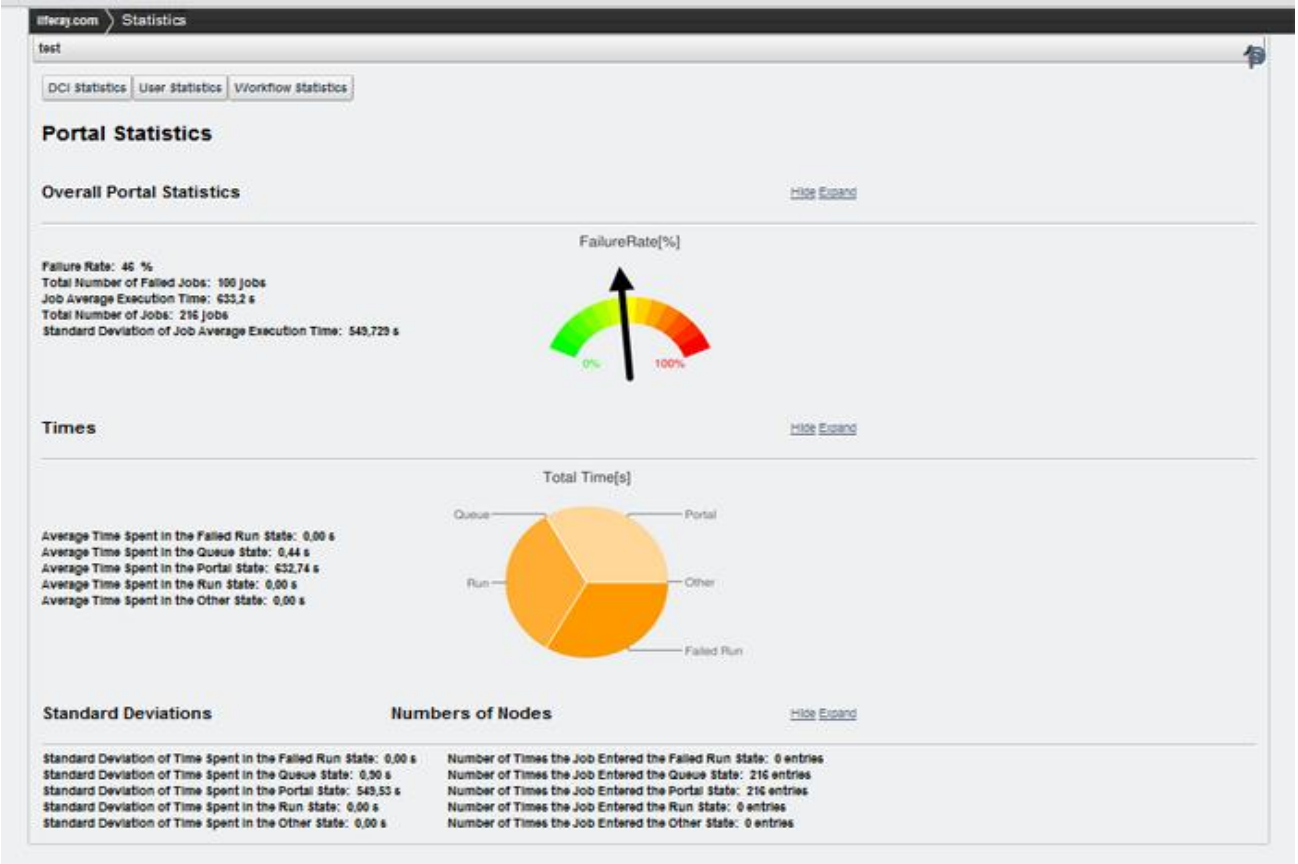

Figure 18 Final Product

Figure 18 shows the portal metrics and serves as the front page to the portlet. The other pages are lain out in the same manner.

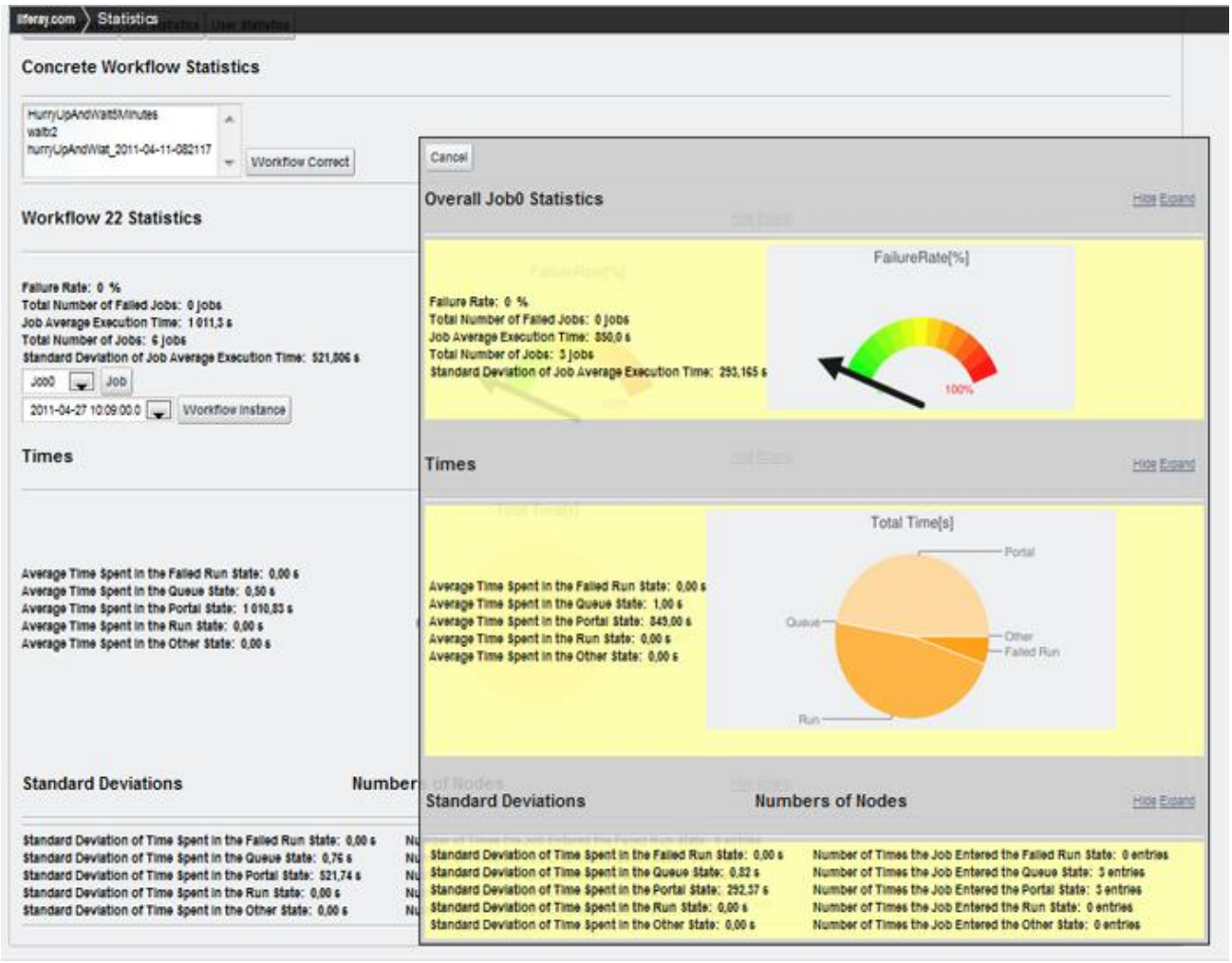

Figure 19 Final Product Concrete Workflow and Abstract Job Metrics

Figure 19 shows an example of an abstract job within a concrete workflow. The abstract job is displayed in a pop up window and the concrete workflow is underneath it.

3.5: Configuration

In order to remove configuration constants from our code, we employed a java configuration file for both the portlet and the calculator service. This file contained information about the database connection and any constants that we wanted to be simple to change. This is advantageous as it makes it simpler to change some of the behavior of the system.

4: Testing

Our testing approach was a combination of iterative and cumulative tests. As we progressed through the implementation of our system we had many smaller components that could be tested individually, which was accomplished as we progressed through the implementation. We also had a dedicated time set aside for testing, which focused on functionality, integration and performance testing. This approach was beneficial as it lent itself to the concurrent development model of the back end and the portlets. All of our tests were executed on our development virtual machine.

For this project the tests consisted of manual testing. Due to the bulk of the functionality being the database interactions, it was simpler to test the functionality manually or with test scripts. While we did consider building a Java database test harness for our database code, we deemed it unnecessarily time consuming.

Our testing also relied heavily on the logging provided by Apache Tomcat's logging system and log file `catalina.out`. This system allowed us to print debugging messages to determine the state of the program when it was running on our development environment instead of our local machines.

4.1: Backend Testing

Throughout the development of the database component's SQL stored procedures and database triggers continuous testing was done in the form of SQL scripts to simulate a workflow running on the grid. Further testing was provided through executing actual workflows on our development portal to test our system with actual data. The final pass for database testing was a suite of testing SQL scripts that tested the behavior of the database programs in a manner similar to unit tests.

The calculator service testing methodology was almost entirely made up of functionality tests, running a workflow and confirming that all of the statistics are correct. There were several types of workflows that we used in order to do this testing. First was a very simple workflow that just executed one job which simply waited for a short period of time. This allowed us to quickly test that the data was propagating through the system. We would then manually confirm the values though comparison with the original data.

4.2: Portlet Testing

Testing for the portlet consisted of making sure it could handle different information loads as well as operate in the expected way. The first part of testing consisted of testing extreme data, both large and small numbers as well as having no data. This ensured that the display would never fail and even if there was no data it would still work.

We also had to test the functionality. This involved making sure every button and selection acted in the way it was supposed to. Furthermore it was tested on multiple browsers to ensure the portlet worked the same way on each browser.

4.3: Functionality Testing

In order to show that our system was working as expected, we ran a suite of functionality tests. The goal of these tests was to explore the behavior of the system at a high level. These tests consisted of workflows that would be executed on the portal and after the execution was complete we viewed the statistics pertinent to the workflow.

There were several workflows that were created for these tests. As the edge cases to our system were all related to parameter studies, and because in WS-PGRADE a non-parameter study workflow is just a parameter study workflow in which all the jobs only execute once, all of the workflows behaved as a parameter study. The most specialized workflow was a workflow that would contain jobs that failed; this tested the behavior of our failure rate statistic. Other workflows were created to test the running statistics of our system. The workflow would run a large number of jobs that had a predictable execution time. We were then able to compare the calculated running times with the expected.

Table 2 Expected Average Running Time Compared to Reported

Concrete Workflow Name	Expected Average Running Time Per Job	Reported Running Actual Time Per Job
QuickLongRunner	10 seconds	345.33 seconds
LongRunner	60 seconds	297.55 seconds
LongRunner_10minEach	600 seconds	1328.7 seconds

The discrepancy between the expected and the actual running time is due to the service that populates the `stat_running` table. Currently, that service does not distinguish between a job instance waiting in the queue of a resource and the job instance being executed on that resource. This was discovered during the implementation of our system. We therefore were careful to show that once the `stat_running` table is populated correctly our system would return the correct values.

4.4: Database Memory Consumption

As was previously mentioned, the aggregate job structure was designed to reduce the memory consumption of the system.

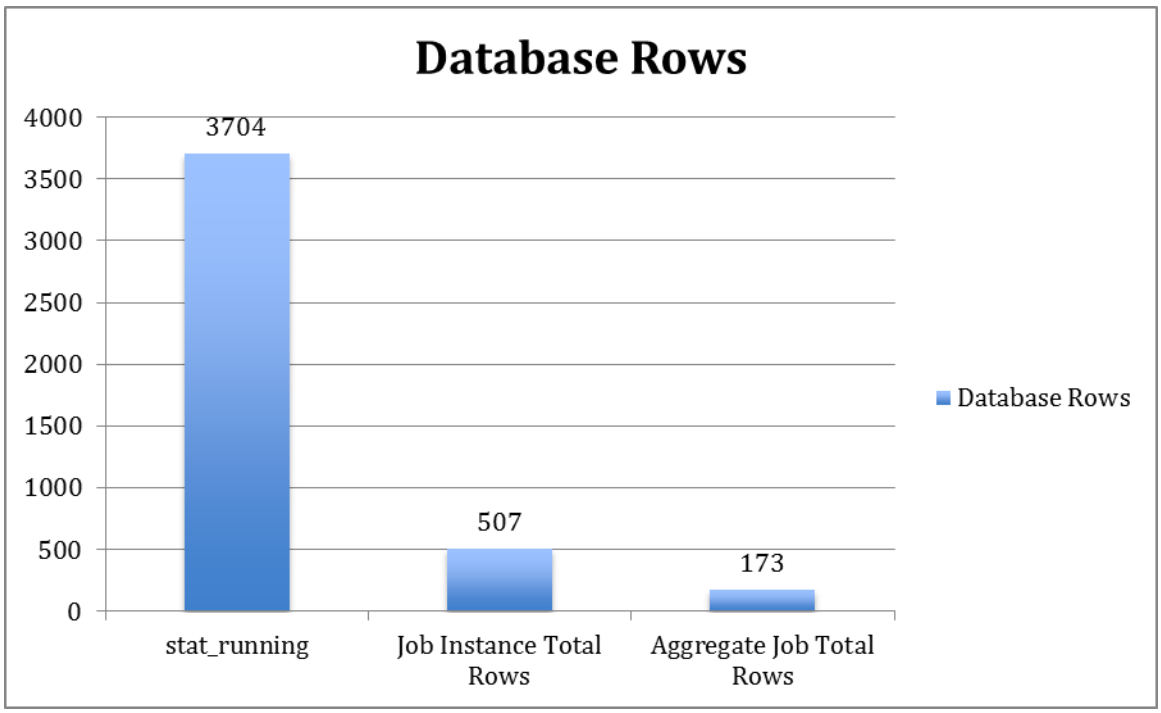

Figure 20 Number of Database Entries for a Workflow

The above graph shows how much this structure reduces the number of database rows as the jobs progress through our system. This data is from a workflow that was executed on our development portal accessing production grids. The workflow had 6 abstract jobs and was executed as a parameter study causing a total of 26 job instances to be executed. In total, the workflow took 2687 seconds. As can be seen in the graph, the number of database rows vastly decreases between `stat_running` to `stat_JobInstance` and `stat_JobInstanceStatus`. This drop is primarily due to

stat_running entries being inserted as a function of time and job instances where stat_JobInstance entries are bounded by job instances and the number of state transitions they experience. The drop from stat_JobInstance to stat_AggregateJob is due to two groupings done. First is the grouping of similar job instances into one aggregate job. The second grouping is due to grouping the similar states from stat_JobInstanceStatus, because a job could enter some states an arbitrary number of times, within stat_AggregateJobStatus.

5: Conclusion

In this project we successfully created a system for the collection of usage statistics for integration with the WS-PGRADE Grid Portal. In its current state, the system will be able to track the execution of workflow instances and job instances executing on the grid and store this information in an efficient manner. We also created a useful visualization interface for this data that displays it for several different levels.

5.1: User Interface

The user interface was successfully implemented as an additional portlet for the WS-PGRADE Grid Portal. The statistics portlet had five pages in the end displaying portal, user, DCI, resource and concrete workflows. From the concrete workflow page the user could choose a workflow instance or an abstract job metrics that appear in a pop-up window.

All the pages used a consistent format. At the top of each page is a navigational menu so the user can easily visit each page without having to use the browsers "back" button. On each page the user is able to hide or show the sections of statistics to see. If there is no data available for one of the levels it instead displays "no data available."

5.2: Back End

Our data management and aggregation services are implemented so that once deployed they will be able to track all job instances that are executed on the portal. While complicated, our aggregate job structure aggregates the data into more efficient units while still allowing meaningful comparisons. It was created in a manner that it could be run on an isolated server from the gUSE system, allowing for any performance issues to be addressed separately.

6: Future Work

Throughout the project we created a list of possible features and metrics for our system. However, due to time constraints or complexity, we were unable to implement everything. We want to identify some areas where we feel that future work on our system would be of value. Our suggested enhancements are generally either new features to the system or additional metrics.

6.1: Revised Architecture

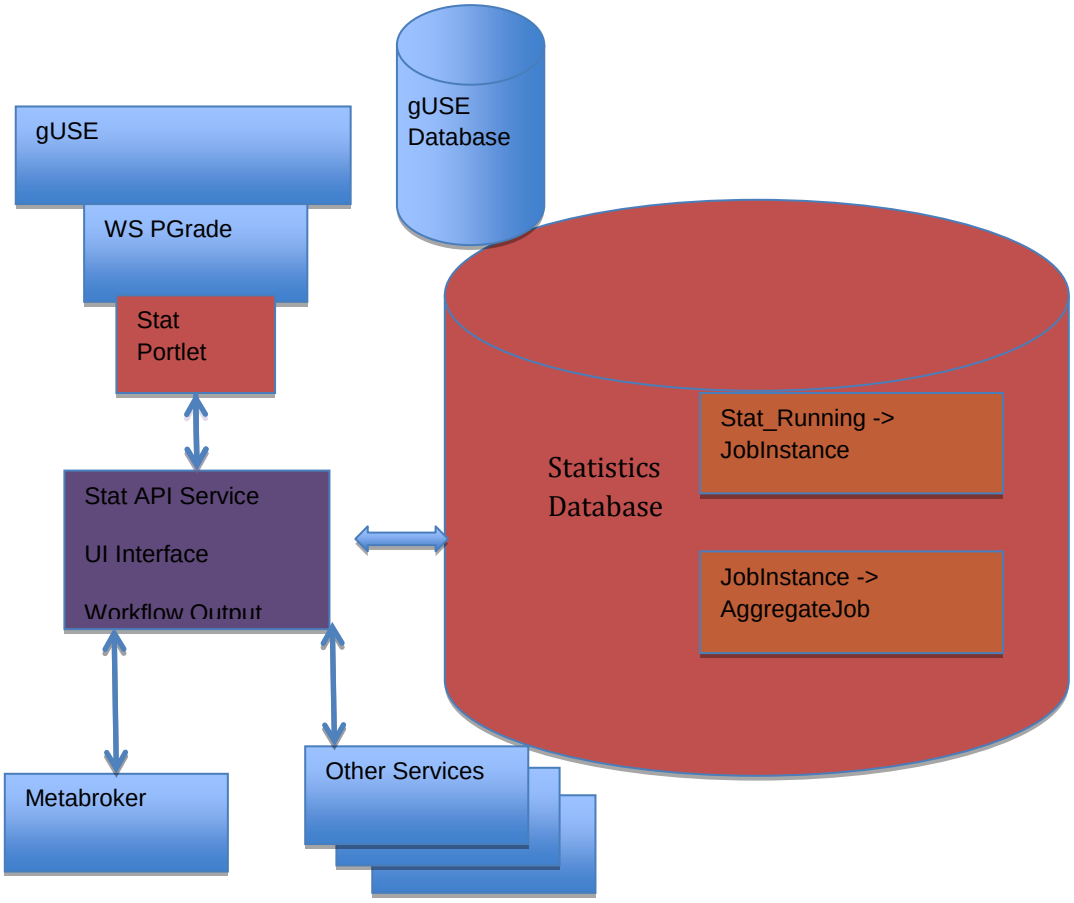

Figure 21 Number of Database Entries for a Workflow

Figure 19 proposes changes to our architecture of the system, with our proposed component in purple. Specifically we would recommend implementing an API service that would replace or add on to the calculator service. This API would provide an access point for the portlet and allow for the possibility of other services to use the statistics data. We

would further suggest to keep the separation of the statistics services from the gUSE services to reduce the impact if at all possible.

6.1.1: Meta-Broker

Assuming the API is implemented, one service that could use our data would be the brokering service. The broker is responsible for assigning job instances to computing resource queues. If an API is implemented, the broker could use the past performance of the job or the resource as part of its decision.

6.1.2: Accounting

Our system can also be used as the first step in an accounting component to gUSE. As previously there was no information being recorded about how the workflows were executing. This component could be used for example, to monetize the portal usage.

6.2: Metrics

Another set of future work would be to expand the set of metrics offered both on the portlet and from an API. Currently, our system does not provide data regarding the current state of the portal, DCI's, or User. Metrics on these categories would be useful, in particular to administrators, to gain knowledge on how the portal or DCI's are being used. Specifically, there is a set of metrics that would be useful about the user that would be available through Liferay. Also, it would be possible to determine how many workflows are currently under submission using the `stat_WorkflowInstance` table. These additional metrics would be best implemented after the API is created as they do not all make sense to be stored in a database.

Another set of possibly useful metrics would be to allow combinations of the current metrics. Currently, it is only possible to view workflow instances individually or combined in the concrete workflow. It could also be feasible for the user to be able to choose a subset of the workflow instances to be combined.

6.3: UI Additions

Some new features could be added to the user interface in later work. First a search function could be added to easily find a concrete workflow, instead of having to find it in a drop down list. Second, the UI could be made more customizable and allow the user to

select which statistics to display. Third, the portlet could display multiple levels of metrics at once, for example display both the portal and the user metrics together. Finally, better navigation techniques could be implemented, for example tabs instead of a menu as well as a “back” or “refresh” button.

7: References

[1] Alejandro Abdelnur, Stefan Hepper. 2003. Java Portlet Specification. October 7, 2003.

[2] Anderson, David P. 2004. BOINC: A System for Public-Resource Computing and Storage. *5th IEEE/ACM International Workshop on Grid Computing .* 2004, pp. 4-10. BOINC PAPER .

[3] gLite. gLite Lightweight Middleware for Grid Computing. [Online] <http://glite.cern.ch/>.

[4] Liferay Inc. What is a Portal? *Liferay Enterprise.Open Source. For Life.* [Online] <https://www.liferay.com/documentation/additional-resources/whitepapers?p_p_id=20&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&_20_redirect=/c/document_library/get_file%3Fuuid%3D8f82e386-3109-4512-bacc-64cda6724751%26groupId%3D14&_20_struts_action=/document_library/file_entry_web_form&_20_fileEntryId=7454189&_20_fileName=What+is+a+Portal%3F.pdf>.

[5] MTA Sztaki LPDS. 2011. P-Grade Grid Portal. [Online] 2011. [Cited: April 7, 2011.] <http://portal.p-grade.hu/>.

[6] *Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal.* **Sipos, Gergely and Kacsuk, Péter. 2005.** 3-4, December 6, 2005, JOURNAL OF GRID COMPUTING, Vol. 3, pp. 221-238.

[7] Oracle. JavaServer Pages Technology. *Oracle.* [Online] [Cited: April 22, 2011.] <http://www.oracle.com/technetwork/java/overview-138580.html>.

[8] *P-GRADE portal family for grid infrastructures.* **Kacsuk, Peter. 2011.** 3, March 10, 2011, Concurrency and Computation: Practice and Experience, Vol. 23, pp. 235-245.

[9] *P-GRADE Portal: A generic workflow system to support user communities.* **Kacsuk, Péter and Farkas, Zoltan. 2011.** 5, May 2011, Future Generation Computer Systems, Vol. 27, pp. 454-465. Arch of PGRADE and basic job and workflow .

[10] SZTAKI LPDS. 2011. Welcome to WS-PGRADE Portal. *GUSE.* [Online] 2011. [Cited: April 22, 2011.]

[11] SZTAKI. MTA SZTAKI Computer and Automation Research Institute Hungarian Academy of Sciences. *The Instute.* [Online] [Cited: 4 1, 2011.] <http://www.sztaki.hu/institute>.

[12] SZTAKI. People - Kacsuk Péter. *SZTAKI.* [Online] [Cited: 4 1, 2011.] <http://www.sztaki.hu/people/008001429/>.

[13] *WS-PGRADE: Supporting parameter sweep applications in workflows.* **Kacsuk, P., Karoczkai, K., Hermann, G., Sipos, G., Kovacs, J.** Nov 2008, Workflows in Support of Large-Scale Science. doi:10.1109/WORKS.2008.4723955.

[14] Google. Coogle Chart Tools/Imgage Charts(aka Chart API). *Google Code.* [Online] 2011. [Cited: 4 11, 2011.] <http://code.google.com/apis/chart/docs/making_charts.html>.

Glossary

Abstract Job

Refers to a job in a Concrete Workflow.

Abstract Job Statistics

Refers to the statistics of all the job instances of the specified Abstract Job aggregated across workflow instances.

Aggregate Job

Aggregation of all job instances that share the same workflow instance, resource and job name.

Concrete Workflow

A workflow that is configured for execution

Concrete Workflow Statistics

Refers to the statistics of all the executions (Workflow Instance) of the specified Concrete Workflow.

DCI

Distributed Computing Infrastructure, a collection of virtual organizations that from which computing resources can be accessed.

DCI Statistics

Refers to the statistics of all jobs and workflows executed on the given DCI.

Google Chart Tools

API used to generate diagrams from the statistics.

Job Instance

A job that is executed on the grid.

Level Of Statistics

Portal, DCI, Resource, User, Concrete Workflow, Workflow Instance, Abstract Job. Refers to what it is possible for the user to view statistics on.

Portal Statistics

Refers to the statistics of all jobs and workflows executed using the instance of the WS-PGRADE Grid Portal.

Resource

A single computing resource. A set of these make up a DCI.

Resource Statistics

Refers to the statistics of all jobs and workflows executed on the given Resource queue.

Stored Procedure (SPROC)

Executable database code that is stored in and run on the database.

Trigger

Executable database code that is automatically executed on some database event such as the insertion into a table.

User

The user that is using the portal, or the user that is interacting with our system.

User Statistics

Refers to the statistics of all jobs and workflow executed by a given user.

Workflow Instance

A single execution of a Concrete Workflow.

Workflow Instance Statistics

Refers to the statistics of all the job instances that were executed for this workflow instance. Also provides the time of execution overall.

Appendix A: Job State Table

This is a table of the possible states that a job instance can enter on the portal or on a resource. Of particular note is our grouping of them as shown in the state type column, which was discussed in the paper. If it becomes necessary to change any of these values or add additional values you must change the enumeration that is in the calculator service project, StatAggregator.jobState.JobState. If it becomes necessary to add terminal states you also have to change the ToJobInstance trigger on stat_running.

Also, this table is subject to change as control of some of the states is given to the grid middlewares. Also, note state 55 which currently only exists in our system to represent the final running state the produced results.

Table 3 Job States

Name	Identifier	Terminal	State Type Assignment
INIT	1	false	StateType.PORTAL
SUBMITTED	2	false	StateType.QUEUE
WAITING	3	false	StateType.QUEUE
SCHEDULED	4	false	StateType.QUEUE
RUNNING	5	false	StateType.RUN
FINISHED	6	true	StateType.TERMINAL
ERROR	7	true	StateType.FAIL
NO_FREE_SERVICE	8	false	StateType.PORTAL
DONE	9	true	StateType.TERMINAL
READY	10	false	StateType.QUEUE
CANCELLED	11	true	StateType.TERMINAL

CLEARED	12	false	StateType.OTHER
PENDING	13	false	StateType.OTHER
ACTIVE	14	false	StateType.OTHER
SUSPENDED	16	false	StateType.PORTAL
UNSUBMITTED	17	true	StateType.TERMINAL
STAGE_IN	18	false	StateType.OTHER
STAGE_OUT	19	false	StateType.OTHER
UNKNOWN_STATUS	20	false	StateType.OTHER
TERM_IS_FALSE	21	true	StateType.FAIL
NO_INPUT	25	false	StateType.FAIL
CANNOT_BE_RUN	99	true	StateType.FAIL
SUCCESS_RUN	55	false	StateType.SUCCESSRUN

Appendix B: Class Diagrams

Appendix B.1 : Calculator Service

This class diagram describe the structure of the calculator service that calculates the statistics based off of the aggregate job data.

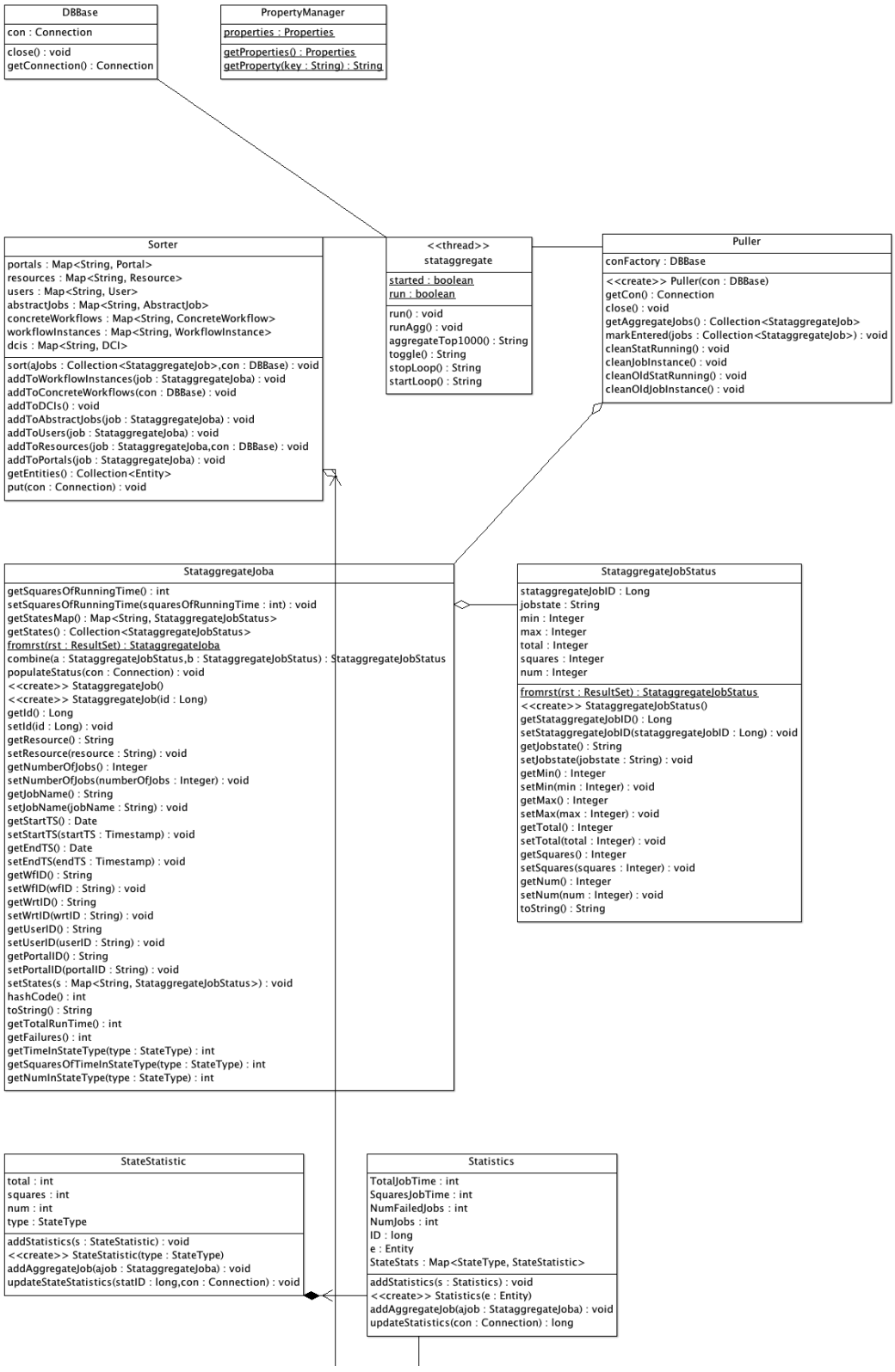

Figure 22 StatAggregator Class Diagram Part 1

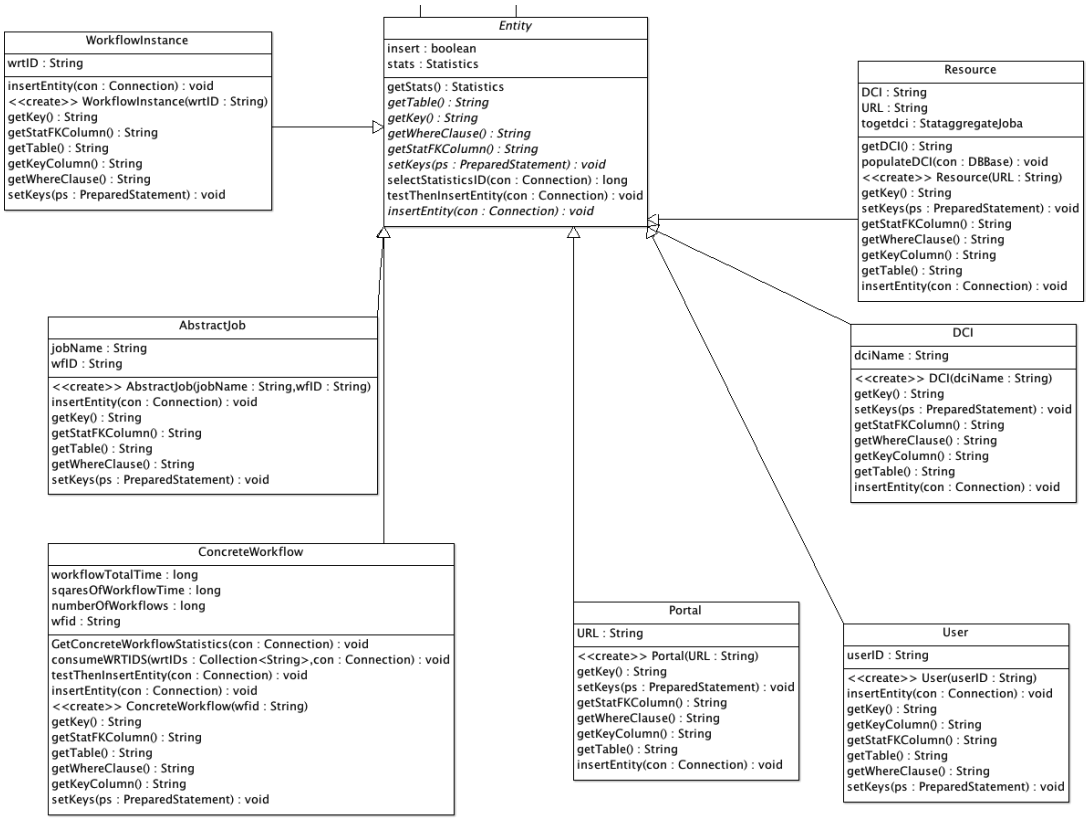

Figure 23 StatAggregator Class Diagram Part 2

Appendix B.2: Portlet Data Access Layer

This diagram describes the structure of the data access layer for the statistics portlet.

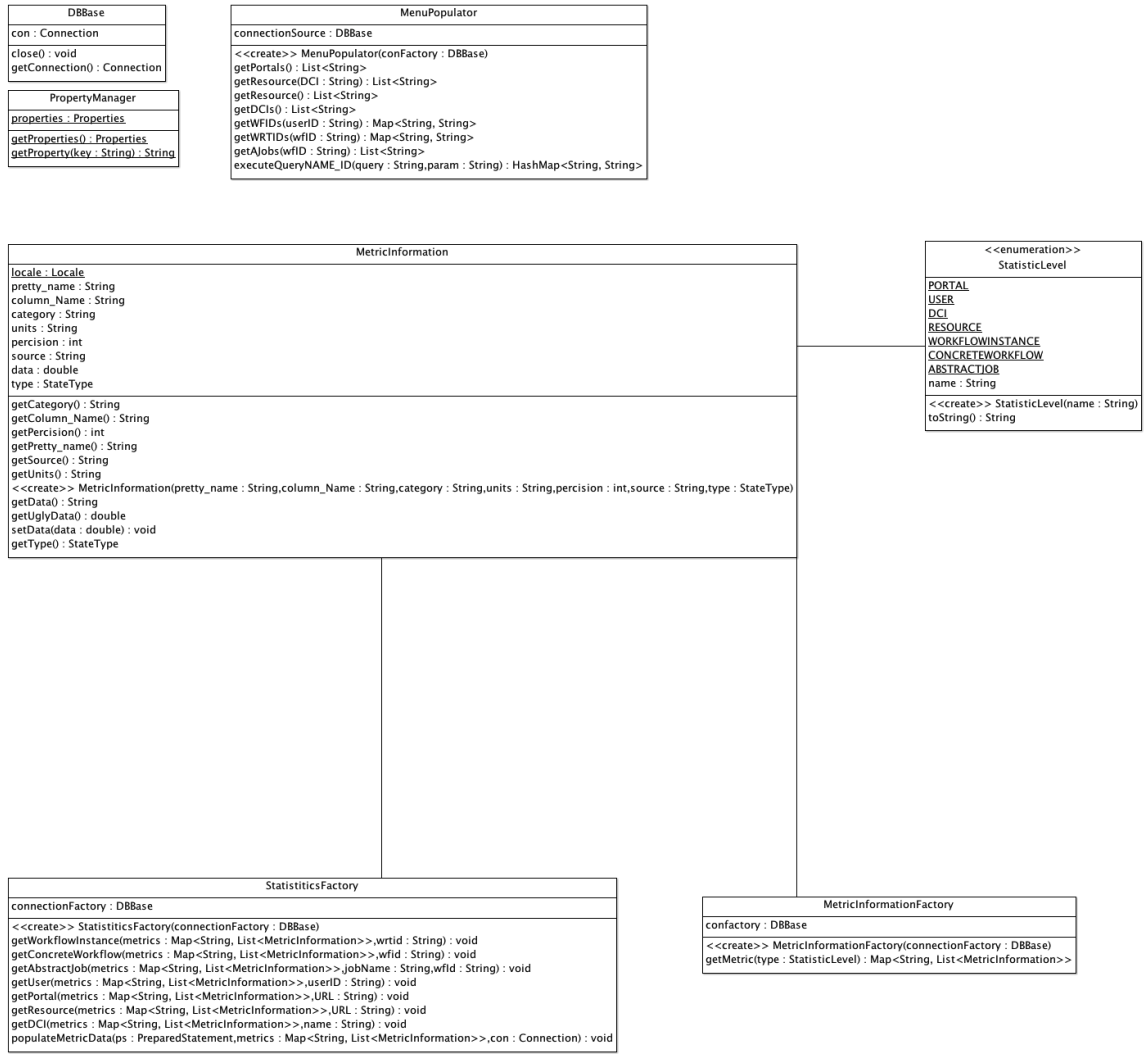

Figure 24 Portlet Data Access Layer Class Diagram

Appendix C: stat_metric_description Table

This table describes the presentation of the metrics we make available to the user on the portlet.

Table 4 stat_metric_description table

Column Name	Pretty Name	Category	Units	Percision	Source Table	for level	State Type	ID
Average	Job Average Execution Time	3	s	1	stat_statistics	all	NULL	2
delta	Workflow Instance Execution Time	6	s	1	stat_WorkflowInstance	workflowinstance	NULL	3
FailureRate	Failure Rate	1	%	2	stat_statistics	all	NULL	4
NumFailedJobs	Total Number of Failed Jobs	1	jobs	0	stat_statistics	all	NULL	5
NumJobs	Total Number of Jobs	3	jobs	0	stat_statistics	all	NULL	6
StdDev	Standard Deviation of Job Average Execution Time	3	s	3	stat_statistics	all	NULL	7
TotalJobTime	Total Running Time	0	s	1	stat_statistics	all	NULL	8
Average	Average Time Spent in the Failed Run State	2	s	2	stat_JobStateTypeStatistics	all	RUN	10
Average	Average Time Spent in the Queue State	2	s	2	stat_JobStateTypeStatistics	all	QUEUE	11

Average	Average Time Spent in the Portal State	2	s	2	stat_JobStateTypeStatistics	all	PORTAL	12
Average	Average Time Spent in the Terminal State	0	s	2	stat_JobStateTypeStatistics	all	TERMINAL	13
Average	Average Time Spent in the Fail State	0	s	2	stat_JobStateTypeStatistics	all	FAIL	14
Average	Average Time Spent in the Run State	2	s	2	stat_JobStateTypeStatistics	all	SUCCESSRUN	15
Average	Average Time Spent in the Other State	2	s	2	stat_JobStateTypeStatistics	all	OTHER	16
StdDev	Standard Deviation of Time Spent in the Failed Run State	4	s	2	stat_JobStateTypeStatistics	all	RUN	17
StdDev	Standard Deviation of Time Spent in the Queue State	4	s	2	stat_JobStateTypeStatistics	all	QUEUE	18
StdDev	Standard Deviation of Time Spent in the Portal State	4	s	2	stat_JobStateTypeStatistics	all	PORTAL	19
StdDev	Standard Deviation of Time Spent in the Terminal State	0	s	2	stat_JobStateTypeStatistics	all	TERMINAL	20
StdDev	Standard Deviation of Time Spent in the Fail State	0	s	2	stat_JobStateTypeStatistics	all	FAIL	21
StdDev	Standard Deviation of Time Spent in the Run State	4	s	2	stat_JobStateTypeStatistics	all	SUCCESSRUN	22
StdDev	Standard Deviation of	4	s	2	stat_JobStateT	all	OTHER	23

	Time Spent in the Other State				ypeStatistics			
Num	Number of Times the Job Entered the Failed Run State	5	entries	0	stat_JobStateTypeStatistics	all	RUN	24
Num	Number of Times the Job Entered the Queue State	5	entries	0	stat_JobStateTypeStatistics	all	QUEUE	25
Num	Number of Times the Job Entered the Portal State	5	entries	0	stat_JobStateTypeStatistics	all	PORTAL	26
Num	Number of Times the Job Entered the Terminal State	0	entries	0	stat_JobStateTypeStatistics	all	TERMINAL	27
Num	Number of Times the Job Entered the Fail State	0	entries	0	stat_JobStateTypeStatistics	all	FAIL	28
Num	Number of Times the Job Entered the Run State	5	entries	0	stat_JobStateTypeStatistics	all	SUCCESSRUN	29
Num	Number of Times the Job Entered the Other State	5	entries	0	stat_JobStateTypeStatistics	all	OTHER	30
FailureRate	Failure Rate	8	%	0	stat_statistics	all	NULL	31
TotalTimeInStates	Failed Run	7	s	2	stat_JobStateTypeStatistics	all	RUN	34
TotalTimeInStates	Run	7	s	2	stat_JobStateTypeStatistics	all	SUCCESSRUN	35

TotalTimeInStates	Queue	7	s	2	stat_JobStateTypeStatistics	all	QUEUE	36
TotalTimeInStates	Portal	7	s	2	stat_JobStateTypeStatistics	all	PORTAL	37
TotalTimeInStates	Terminal	0	s	2	stat_JobStateTypeStatistics	all	TERMINAL	38
TotalTimeInStates	Fail	0	s	2	stat_JobStateTypeStatistics	all	FAIL	39
TotalTimeInStates	Other	7	s	2	stat_JobStateTypeStatistics	all	OTHER	40
average	Average Workflow Execution Time	6	s	2	stat_ConcreteWorkflow	concretewworkflow	NULL	41

Appendix D: Installation Manual

In order to deploy the statistics system there are three components that must be deployed.

Appendix D.1: Database Deployment

To modify the database with our schema changes, please run the provided scripts:

- GUSE_stat_statistics.sql
- GUSE_stat_running.sql
- GUSE_stat_WorkflowInstance.sql
- GUSE_stat_JobInstance.sql
- GUSE_stat_JobInstanceStatus.sql
- GUSE_stat_AggregateJob.sql
- GUSE_stat_AggregateJobStatus.sql
- GUSE_stat_AbstractJob.sql
- GUSE_stat_JobStateTypeStatistics.sql
- GUSE_stat_portal.sql
- GUSE_stat_ConcreteWorkflow.sql
- GUSE_stat_resource.sql
- GUSE_stat_DCI.sql
- GUSE_stat_metric_description.sql
- GUSE_stat_user.sql
- GUSE_routines.sql

Once these scripts are executed, confirm that there are 15 new tables [or 14 if there was already the stat_running table installed, in which case confirm that it was modified]. Also confirm that the following triggers and stored procedures are present.

Triggers

```
BEFORE INSERT ON stat_running

BEFORE UPDATE ON stat_JobInstance      TOAGGJOB

BEFORE UPDATE ON stat_ConcreteWorkflow calculate_workflow_
```

```
BEFORE INSERT ON stat_ConcreteWorkflow calculate_workflow_delta

BEFORE UPDATE ON stat_WorkflowInstance calculate_workflow_delta

BEFORE UPDATE ON stat_statistics calc_statistics_stats_update

BEFORE UPDATE ON stat_JobStateTypeStatistics
calc_statetype_stats_update

BEFORE INSERT ON stat_statistics  calc_statistics_stats_insert

BEFORE INSERT ON stat_JobStateTypeStatistics
calc_statetype_stats_insert
```

Stored Procedures

```
JobInstanceToAggregateJob

CreateOrAddToJobInstance
```

See database description section for a brief description of the use of each of these elements.

It should also be possible for all of these components to be run on a separate database from the gUSE database if deemed pertinent. If so, please make sure that the connection information is changed appropriately. Also, make sure to test the portlet's MenuPopulator.java as it does use some gUSE database tables in order to provide useful names for concrete workflows, jobs, and DCIs.

Appendix D.2: Calculator Deployment

There are several options to deploy the calculator service. It is set up as a web service which can be on the same server or on a distinct server from the portal. First step is to locate the `statAggregator.properties` file and set the values in there for the database connection, how long to wait for non-terminated jobs and stat_running entries, and for the frequency of the poll. Then, install the project as a web service on a server with access to the database with the configuration information given.

Once installed, go to the URL `[SERVER]/StatAggregator` which is currently set up to toggle the polling mechanism of the service. Alternate initialization may be

recommended using the `web.xml` file to set the service to start with the server. See the file `stataggregate.java` and `index.jsp` to see how to start the service if an alternate method is called for.

The calculator service also uses information from the gUSE database. Specifically, it uses it in order to provide a resource URL to DCI name mapping. See `Resource.populateDCI()`.

Appendix D.3: Portlet Deployment

To deploy the portlet first set the values in the configuration file to give database access to the database where the statistics data is being stored and to set the locale and language (defaults to Hungary and Hungarian). The configuration file also requires the URL of the portal. The language and locale is used for formatting of the values on the portlet.

To deploy the portlet on Liferay go to the manage tab at the top of the page and select control panel. At the bottom of the list under server choose Plugins Installation. Under the Plugins Installation click the button Install More Portlets and choose Upload File. Select choose file and locate the .war file to be uploaded. Then click Install and wait for the success message to appear.

Appendix D.4: Stopping Statistics

If it becomes necessary to stop the statistics functionality besides reverting the system, the simplest method is to toggle off the StatAggregator using the URL `[SERVER]/StatAggregator toggle` and drop the trigger on the `stat_running` table. This will prevent data from progressing through the system and will stop the polling mechanism of the calculator service.

Appendix E: Database Description

Table 5 Database Table Descriptions

Table Name	Description
stat_running	Intermediate Data many entries/job instance - supplied by gUSE, modifications: entered column, default 0, when 1 - delete, database trigger
stat_JobInstance	Intermediate Data one entry/job instance
stat_JobInstanceStatus	Intermediate Data one entry per job state transition
stat_AggregateJob	One entry combining all JobInstance with same JobName, Resource, wrtID
stat_AggregateJobStatus	One entry per job state visited by any of the job instances combined into this
stat_WorkflowInstance	One entry per workflow instance
stat_AbstractJob	One entry per job in the DAG of a concrete workflow
stat_ConcreteWorkflow	One entry per concrete workflow
stat_user	One entry per user
stat_WorkflowInstance	One entry per workflow instance
stat_portal	One entry per portal
stat_DCI	One entry per DCI
stat_resource	One entry per resource queue
stat_statistics	Contains calculated statistics about jobs. One entry for each row in stat_user/portal/DCI/resource/AbstractJob/ConcreteWorkfl

	ow/WorkflowInstance
stat_JobStateTypeStatistics	Contains calculated statistics about job states
stat_metric_description	Contains information about the access and grouping of metrics for display

Table 6 Database Tigger Descriptions

Name	Description
BEFORE INSERT ON stat_running toJobInstance	stat_running entries into stat_JobInstance and stat_JobInstanceStatus entries. Calls CreateOrAddToJobInstance
BEFORE UPDATE ON stat_JobInstance TOAGGJOB	stat_JobInstance and status entries to stat_AggregateJob and stat_AggregateJobStatus. Calls JobInstanceToAggregateJob
BEFORE UPDATE ON stat_ConcreteWorkflow	Calculate average and standard deviation
BEFORE INSERT ON stat_ConcreteWorkflow	Calculate average and standard deviation
BEFORE UPDATE ON stat_WorkflowInstance	Calculate workflow execution time
BEFORE UPDATE ON stat_statistics	Calculate average and standard deviation
BEFORE UPDATE ON stat_JobStateTypeStatistics	Calculate average and standard deviation
BEFORE INSERT ON stat_statistics	Calculate average and standard deviation
BEFORE INSERT ON stat_JobStateTypeStatistics	Calculate average and standard deviation

Table 7 Stored Procedures

Name	Description
JobInstanceToAggregateJob	Chooses to create new stat_AggregateJob entry or updates an existing one. Also inserts or updates stat_AggregateJobStatus with appropriate data from stat_JobInstanceStatus
CreateOrAddToJobInstance	Adds stat_running data to a stat_JobInstance row [or creates one if does not already exist]

Appendix F: User Manual

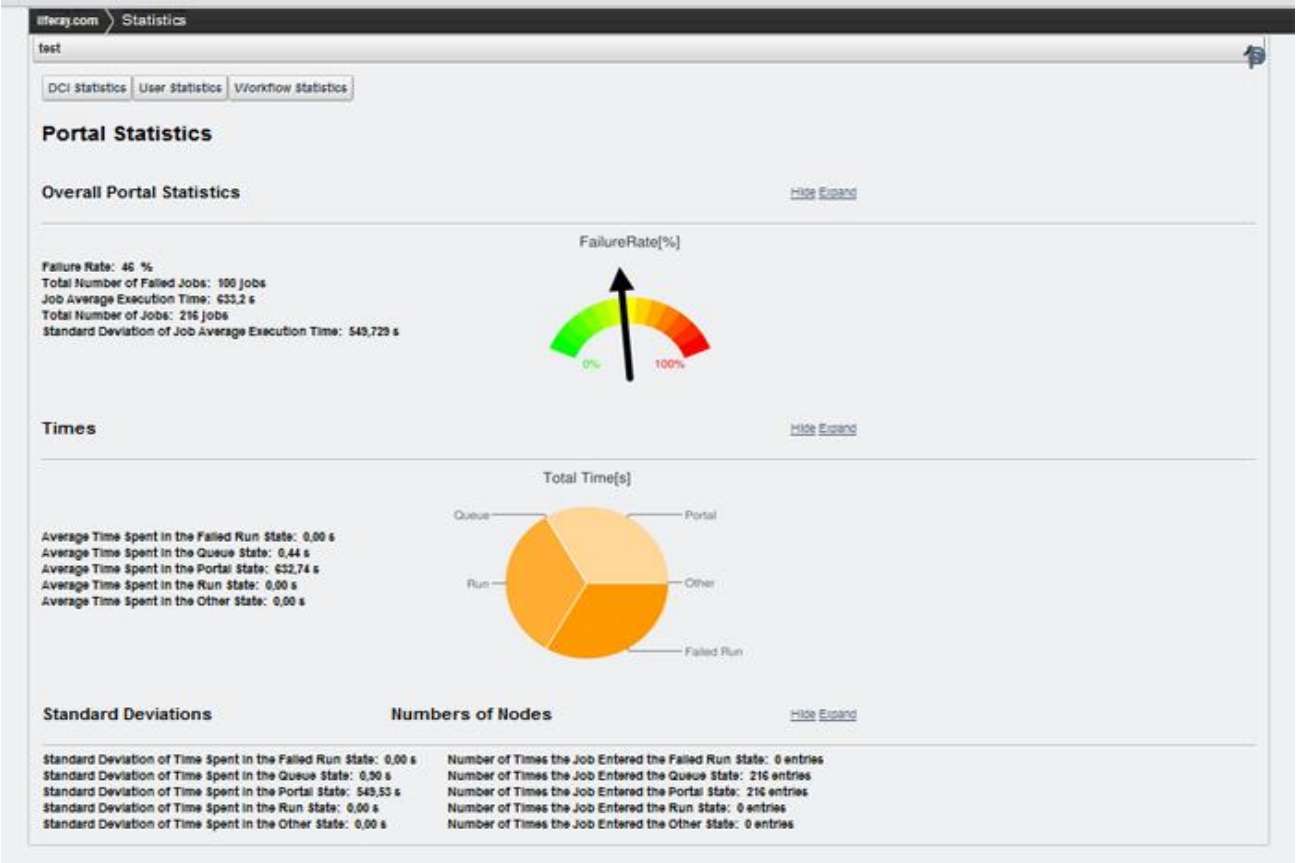

Figure 25 User Interface

F.1: Introduction

The aim of the statistics portlet is to allow users to view metrics on seven levels, portal, user, DCI, resource, concrete workflow, workflow instance and abstract job. This is accomplished by allowing users to navigate to different pages to see the level of statistics they want. The statistics portlet is an addition to the pre-existing portlets on the WS-PGRADE Grid Portal. The default page view upon clicking the statistics tab is the metrics for the portal. The other pages can be accessed through a menu at the top of the page. For any section a user can choose to expand and minimize the amount of data they wish to see by clicking on “expand” or “hide”. Descriptions and usage information can be found below.

F.2: DCI Metrics

Figure 26 Selecting DCI Statistics

To navigate to DCI metrics the user clicks the DCI menu button at the top of any page. Once on the DCI page the user chooses from a drop down list of available DCI's. Once chosen, the user clicks the "DCI" button next to it and the metrics will be displayed. The DCI metrics can be useful for comparing different DCI's and checking performance.

F.3: Resource Metrics

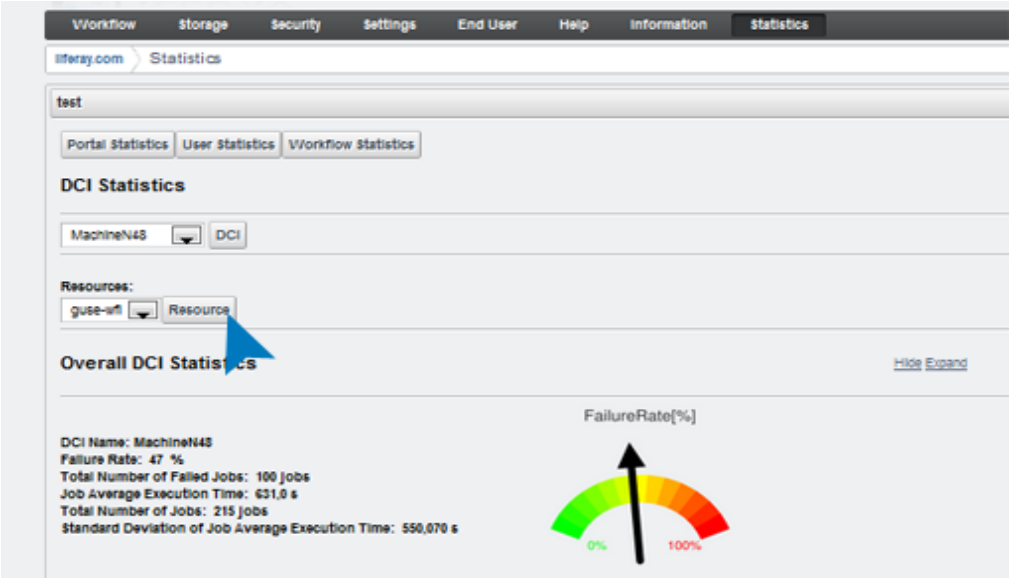

Figure 27 Selecting Resource

To navigate to resource metrics the user will need to have already choosen a DCI. Once a DCI is chosen, a new dropdown list of available resources on that DCI will become available. The user can choose the resource they wish to view.

F.4: User Metrics

To navigate to user metrics the user clicks the User button at the top of any page, once clicked they will be directed to a page with all of the current users metrics.

F.5: Concrete Workflow Metrics

To navigate to concrete workflow metrics the user selects the Concrete Workflow button at the top of any page. Once the choice is made the user will need to choose which of their concrete workflows they wish to view. They can select up to three to view at a time by holding the shift or ctrl keys when selecting multiple. The statistics will be displayed below once the user clicks the button to the right of the selection menu, in order that the concrete workflows appear in the list.

F.6: Workflow Instance and Abstract Job Metrics

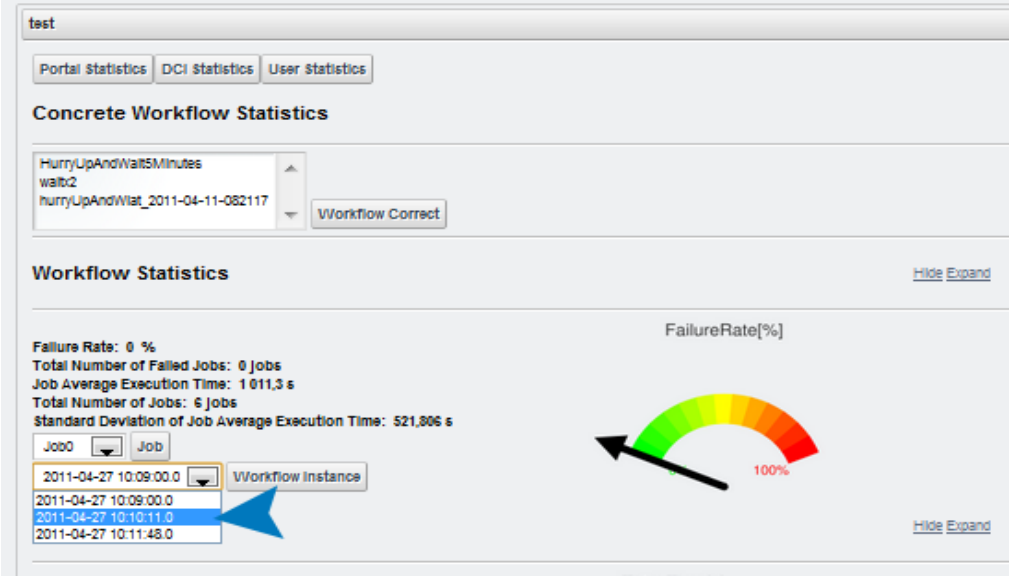

Figure 28 Concrete Workflow Metrics

To view metrics on workflow instances or abstract jobs, the user must have first chosen a concrete workflow. Once a concrete workflow is selected two drop down menus of available workflow Instances and abstract jobs will appear for each concrete workflow selected. The user will choose one to display and the metrics will appear in a pop-up window.

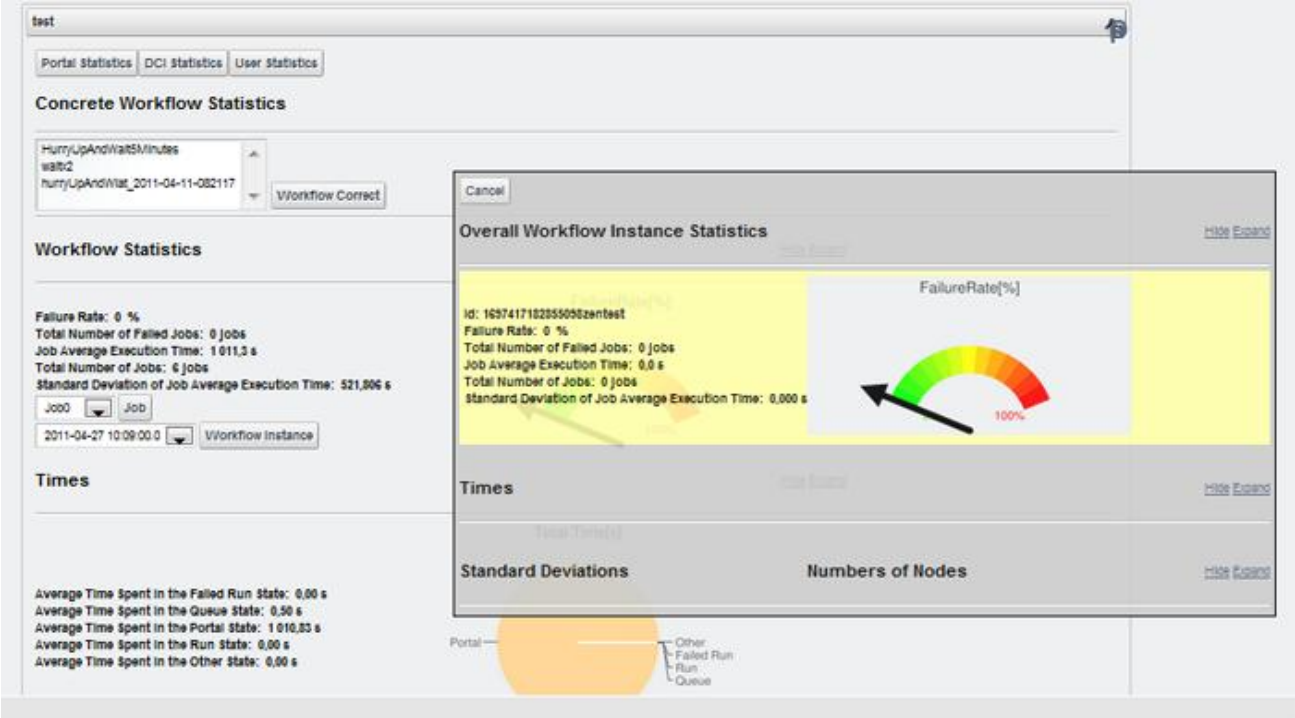

Figure 29 Pop Up Window for Workflow Instance