<myJAM/> – Accounting und Monitoring auf Rechenclustern

Stephan Raub, Dennis-Bendert Schramm, Stephan Olbrich

Lehrstuhl für IT-Management, Institut für Informatik /
Zentrum für Informations- und Medientechnologie (ZIM)
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1
40225 Düsseldorf
raub@uni-duesseldorf.de
dennis-bendert.schramm@uni-duesseldorf.de
olbrich@uni-duesseldorf.de

Abstract: Im Rahmen einer Kooperation der Heinrich-Heine-Universität Düsseldorf (HHU) mit der Bull GmbH auf dem Gebiet des High Performance Computing werden Anforderungen an das homogene Management von heterogenen Clusterkonfigurationen analysiert und neue Lösungsansätze entwickelt. Ein Ergebnis des Projekts wird im Papier dargestellt: die prototypische Implementierung eines netzverteilten, modularen, portablen und über eine Weboberfläche interaktiv nutzbaren Systems für das Accounting und Monitoring auf Rechenclustern mit integriertem Projekt- und Anwendungsmanagement. Das Werkzeug *<myJAM/>* („Job Accounting und Monitoring“, `www.myjam.uni-duesseldorf.de`) unterstützt bereits den batchbasierten Betrieb des zentralen heterogenen Linux-Rechenclusters an der HHU unter Linux mit einer Anbindung an das Batchsystem PBS Pro und soll mittelfristig als plattformübergreifende Open-Source-Software bereit gestellt werden.

1 Einleitung

In den letzten Jahren haben sich für das Hochleistungsrechnen Cluster-Lösungen stark verbreitet, die auf kostengünstigen Komponenten der Massenproduktion basieren. Über Interprozess- und Managementnetzwerke wird sowohl die Parallelisierung von Anwendungen als auch ein koordinierter Betrieb unterstützt. Während Cluster in der Vergangenheit meist auf einem homogenen Design vieler gleichartigen Rechenknoten basierten, besteht inzwischen zunehmend der Bedarf, verschiedenartige Rechnerarchitekturen zu integrieren und somit unter einer homogenen Betriebs- und Anwendungsumgebung wahlweise heterogene Systemkomponenten zu nutzen. Darüber hinaus kommen Hybridrechner zum Einsatz, in denen herkömmliche Prozessoren mit Spezialprozessoren, z. B. Compute-Beschleunigern wie Nvidia Tesla GPU-Server, kombiniert werden.

Um die Anwendungen optimal über die Ressourcen des Clusters zu verteilen, werden in der Regel Batchsysteme genutzt. Diese abstrahieren alle technischen Details, wie z. B. das

Finden geeigneter freier Knoten, auf diesen die Anwendungen zu starten, dann zu überwachen, etc. So ist ein Detailwissen über diese komplexen Prozesse für die Nutzer eines Clusters nicht zwingend notwendig, da ein solches Batchsystem eine uniforme Sicht auf den Cluster bietet. Darüber hinaus werden über ein Batchsystem site-spezifische Regeln (die sog. Policies) etabliert, wie z. B. user- oder gruppenspezifische Beschränkungen der kumulativ genutzten Ressourcen. In der Umkehrung können anderen Gruppen exklusive Nutzungsrechte über Ressourcen gewährt werden.

Allerdings werden noch komfortable Werkzeuge benötigt, die mittels Accounting und Monitoring zum effizienten Betrieb, zur Analyse der Nutzung und zur proaktiven Unterstützung der Anwendungsprojekte sowie gegebenenfalls auch zur Abrechnung beitragen.

1.1 Accounting

Unter dem Schlagwort *Accounting* verstehen wir hier die Erfassung sowie die Nutzer-, Job- bzw. Projekt-bezogene Zuordnung der genutzten Ressourcen als Grundlage für eine Berechnung der Kosten der erbrachten IT-Leistungen. Dabei kann es sich auch um rein *virtuelle Kosten* handeln, die alleine für statistische Zwecken erhoben werden, z. B. um die Ressourcennutzung von verschiedenen Institutionen, die einen Cluster gemeinsam nutzen, quantifizieren zu können. Neben der Abrechnungsmöglichkeit stellen auch die Ressourcenplanung und der Einfluss auf die Kontingentierung Motivationen dar.

1.2 Monitoring

In Rahmen des Projekts verstehen wir unter *Monitoring* die unmittelbare systematische Erfassung und Visualisierung der Ressourcennutzung auf einem Cluster. Monitoring ist ein Schlüsselelement, um eine Datenbasis für eine Bewertung und nötigenfalls eine Optimierung der Effizienz eines Systems zu haben. Auch um das Greifen von Policies validieren zu können, ist ein detailliertes und aussagekräftiges Monitoring unerlässlich.

2 Analyse der Anforderungen und Defizite verfügbarer Tools

Zur Unterstützung des Betriebs von Rechenclustern sowie der Anwendungen werden Accounting- und Monitoring-Werkzeuge benötigt, die den folgenden Zwecken und Anforderungen genügen: batchorientierte Ressourcenerfassung, optional prozessorientiertes Accounting, Integration heterogener, netzverteilter Konfigurationen, hoher Interaktionsgrad über eine Weboberfläche, Plattformunabhängigkeit bzw. Portierbarkeit sowie zumindest rudimentäre Verwaltung von Projekten bzw. deren Metadaten.

Die verfügbaren Varianten von Monitoring-Tools sind vielfältig: von kostenlosen Open-

Source-Lösungen bis hin zu hochpreisigen kommerziellen Systemen. Diese Tools überwachen zumeist die einzelnen Knoten eines Clusters und zeigen jeweils Daten über Auslastung, Fehler, etc. an – sie bieten also eine Node-basierte Sicht auf den Cluster.

Die meisten Batchsysteme bringen darüber hinaus auch eigene Tools mit, die ein Monitoring auf Grundlage der dem Batchsystem bekannten Daten anbieten. Die Batchsysteme überwachen die einzelnen Jobs und über die Jobs die von diesen genutzten Knoten eines Clusters – sie bieten also eine Job-basierte Sicht auf den Cluster. Die Tools, die mit einem Batchsystem ausgeliefert werden, bieten z. T. zusätzlich auch eine Node-basierte Sicht. Doch arbeiten diese in der Regel auch nur mit „ihrem“ Batchsystem zusammen. Auch auf die Heterogenisierung gehen die verfügbaren Tools bisher nur geringfügig ein.

3 Grundideen zu einem eigenen Lösungsansatz

Aufgrund der Unzulänglichkeiten der für Accounting und Monitoring verfügbaren Werkzeuge im Vergleich mit den Anforderungen im Rahmen des Betriebs heterogener, kontinuierlich sich verändernder Rechencluster wurde eine Eigenentwicklung begonnen.

Ursprünglich war eine Abspaltung und Weiterentwicklung („Fork“) des Open-Source-Projektes „myPBS“ [MYP] geplant, welches jedoch ab Version 0.8.6 vom 05. April 2006 nicht mehr weiterentwickelt worden ist. Mittlerweile wurde mit dem Werkzeug <myJAM/> jedoch eine komplette Neuentwicklung durchgeführt, so dass es mit seinem „Ur-Vater“ myPBS praktisch nichts mehr gemein hat, außer einigen grundsätzlichen Konzepten, wie z. B. die Verwendung von Perl, MySQL und einem Web-Frontend oder einige Terminologien wie die *Service Unit (SU)* oder die Projekte. In myPBS existierten keine klar voneinander getrennten Programmier-Schichten, so dass durch viele interne Abhängigkeiten der Code schwer wartbar vorgefunden wurde. Darüber hinaus war das Web-Frontend nicht interaktiv und benutzte viele veraltete oder proprietäre Bibliotheken. Moderne Web-Konzepte wie AJAX fehlten, und es wurde keinerlei W3C-Standard eingehalten. All diese Mankos wurden im Zuge der Neuentwicklung ausgemerzt.

<myJAM/> versteht sich als Monitoring- und Accounting-Tool, das mit einem oder mehreren – gegebenenfalls auch verschiedenen – Batchsystemen zusammen arbeitet. Es führt alle bereits verfügbaren und selbst ermittelten bzw. eingegebenen Informationen über Jobs und Knoten, die es vom Batch- oder Betriebssystem bekommt, bzw. Angaben zu den Projekten in Echtzeit zusammen und speichert diese in einer Datenbank. Von dort aus können sowohl Echtzeit- als auch historische Analysen und Darstellungen nach verschiedensten Kriterien über ein hoch interaktives Web-Frontend abgerufen werden. Unschätzbarer Vorteil dieses webbasierten Frontends ist die clientseitige Unabhängigkeit vom verwendeten Betriebssystem. Sowohl homogene als auch beliebig heterogene Cluster werden nativ unterstützt.

Durch das lokale Echtzeit-Monitoring unterscheidet sich <myJAM/> von vielen bereits verfügbaren Lösungen (auch kommerziellen), die lediglich eine Analyse bestehender Log-Files *a posteriori* durchführen. Es ist eine nahe liegende Idee, für das Monitoring Informationen des Batchsystems zu nutzen, wenn ein HPC-Cluster ausschließlich darüber genutzt

wird. Das Batchsystem „weiß“, welche Jobs gerade wo laufen und wie viele Ressourcen von diesen Jobs genutzt werden. Somit liegen schon detaillierte Informationen über die aktuelle Cluster-Auslastung vor und müssen nicht erst aufwändig erneut ermittelt werden.

Das Accounting funktioniert in <myJAM/> projektbasiert. Ein Projekt kann mehrere User enthalten und ein User kann mehreren Projekten angehören. Jedem Projekt kann detailliert der Zugang zu beliebigen Queues des Batchsystems gewährt werden

<myJAM/> macht sich ausschließlich Open-Source-Softwaretechnologien wie PHP [PHP], Perl, Apache [Apa], MySQL [SQL] und OpenFlashChart [Gla] zu Nutze.

Obwohl alle bisherigen <myJAM/>-Installationen unter Linux betrieben werden, sind die genannten Software-Komponenten auch für Windows und MacOS verfügbar. <myJAM/> arbeitet jedoch aktuell nur mit Linux-Batchsystemen (wie z. B. PBS(Pro) [PBS07] oder Torque [TRQ]) zusammen. Auch <myJAM/> wird mittelfristig als Open-Source-Software bereit gestellt.

4 Technische Realisierung: <myJAM/>

<myJAM/> besteht aus drei Hauptkomponenten (siehe Abbildung 1):

- Die <myJAM/>-Datenbank: Eine MySQL-Datenbank, in der alle Informationen über Projekte, User, Jobs und Anwendungen gespeichert werden.
- Der <myJAM/>-Daemon: Ein *nix-Daemon, der auf jedem PBS-Serverknoten läuft. Er sammelt die Informationen über laufende und wartende Jobs vom Batchsystem oder vom Betriebssystem und speichert sie in der Datenbank. Darüber hinaus werden die laufenden Anwendungen klassifiziert.
- Das <myJAM/>-Web-Frontend: Eine hoch-interaktive Web-Applikation, mit dem die gesammelten Informationen in der Datenbank nach verschiedensten Kriterien analysiert und visualisiert werden können. Auch User und Projekte können verwaltet werden.

Jede dieser drei Komponenten (siehe Abschnitte 4.2 bis 4.5) läuft unabhängig von den anderen. Die Kommunikation findet ausschließlich über das standardisierte „MySQL Network Protocol“ statt, weshalb jede Komponente auch auf einem eigenen Server laufen kann. Grundsätzlich können mehrere <myJAM/>-Daemons auf verschiedenen Batch-Servern laufen, um mehrere Batchsysteme überwachen zu können.

Durch die Modularität und die wohldefinierte Kommunikation ist es relativ einfach, neue Daemons für weitere Batchsysteme zu entwickeln. Die Datenbank und das Web-Frontend bedürfen keiner Anpassung. Aus dem gleichen Grund könnten auch einfach neue Frontends entwickelt werden, z. B. ein Kommandozeilen-Interface.

Zusätzlich kommen noch *Prolog*- und *Epilog*-Skripte zum Einsatz. Diese Skripte werden von den meisten Batchsystemen unmittelbar vor (Prolog) und nach (Epilog) dem Start

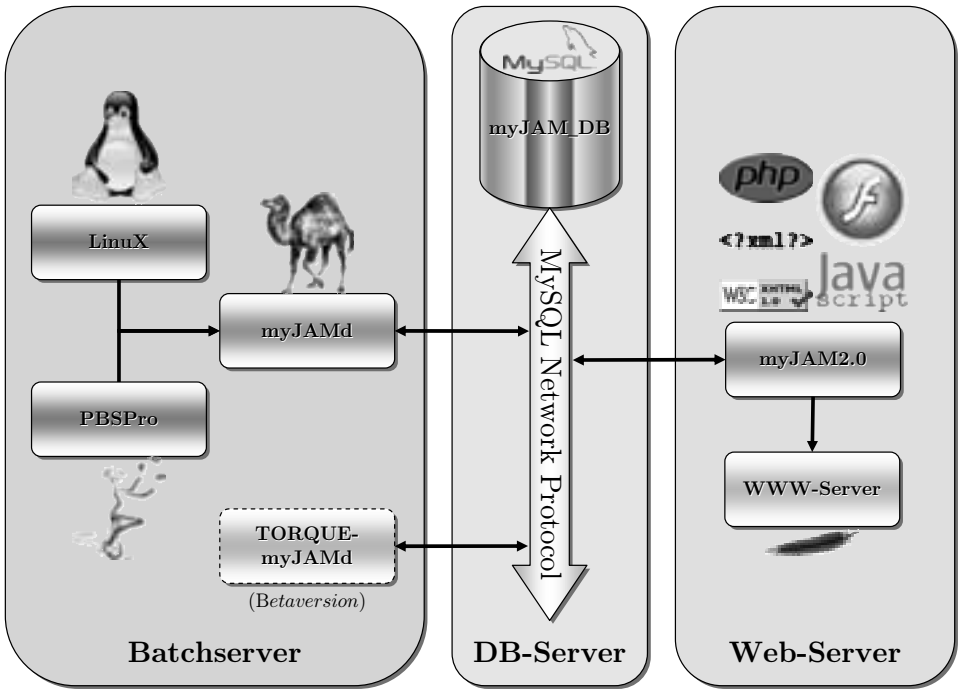

Abbildung 1: Die einzelnen Komponenten von <myJAM/> und deren Interaktionen untereinander

eines Jobs automatisch ausgeführt. <myJAM/> verwendet diesen Mechanismus, um jobbezogene Daten (z. B. Knotennummer, Zeitstempel) in der Datenbank zu speichern.

4.1 Abrechnungsmodelle

<myJAM/> bietet, basierend auf den Accounting-Informationen, die Möglichkeit des *Billings*, also der Berechnung des ökonomischen Wertes der genutzten Ressourcen. Grundeinheit für das Accounting (quasi eine virtuelle Währung innerhalb von <myJAM/>) ist die *ServiceUnit* (*SU*), die einer CPU(Core)-Stunde entspricht. Die Umrechnung der virtuellen Währung *SU* in eine monetäre Währung erfolgt in <myJAM/> über frei definierbare *Kostenmodelle*. Ein Kostenmodell besteht aus zwei Tarifen: dem *Normal-Tarif* und dem *Overrun-Tarif*, jeweils in Währung pro *SU*. Beide Werte können unterschiedlich sein, müssen es aber nicht.

Ein Projekt kann kostenpflichtig oder kostenfrei sein. Ist es kostenpflichtig, können *Credit*- oder *PrePaid*-Kostenmodelle zum Einsatz kommen. Beim Credit-Kostenmodell können User des Projektes *SUs* verbrauchen und nach Beendigung des Projektes oder in regelmäßigen Abständen für die bis dato genutzten *SUs* bezahlen (Normaltarif). Bei PrePaid-

Projekten können User des entsprechenden Projektes nur diejenigen *SUs* verbrauchen, die als *SU*-Guthaben auf dem Konto des Projektes vorhanden sind (durch vorhergehende Einzahlung zum Normaltarif). Darüber hinaus ist es möglich, eine gewisse Überziehung zuzulassen und diese mit der nächsten Einzahlung zu verrechnen (zum Overrun-Tarif), oder eine gesonderte Rechnung dafür auszustellen.

4.2 Die <myJAM/>-Datenbank

Die Datenbank von <myJAM/> wurde mit MySQL5 [SQL] realisiert. Unser Datenbankdesign berücksichtigt weitgehend die *dritte Normalform*, um Dateninkonsistenzen vorzubeugen. Aus Performance-Gründen wurden lediglich einige wenige Tabellen auf die zweite Normalform denormalisiert. User, Projekte, Warteschlangen, Hosts, Anwendungen und Kostenmodelle bilden die zentralen Tabellen. Alle Einträge besitzen Surrogatschlüssel als Primärschlüssel. Attribute, die auf Einträge einer anderen Tabelle referenzieren, benutzen diese Surrogatschlüssel als Fremdschlüssel. Für das Web-Frontend werden die Einträge in den Tabellen und die Referenzen auf andere Tabellen in PHP5-Objekte im Rahmen der <myJAM/>-Klassenbibliothek umgesetzt (*object-relational mapping*).

Mehrere Prozesse greifen auf die Datenbank zu: das <myJAM/>-Web-Frontend, ein oder mehrere <myJAM/>-Daemons und diverse Prolog- und Epilog-Skripte. Daher ist ein rigoroses Locking unbedingt notwendig. Der Nachteil dabei ist, dass jedes Mal, wenn eine oder mehrere Tabellen gelockt sind, alle anderen Prozesse bis zur Freigabe der entsprechenden Tabellen blockiert sind. Zurzeit priorisieren wir absolute Datenkonsistenz, weshalb wir recht ausgiebig vom Locking Gebrauch machen, wohl wissend, welche anderen Probleme (z. B. die Wartezeiten im Web-Frontend) wir damit produzieren. Die Locking-Strategie ist einer der Punkte unserer ständigen Weiterentwicklung.

4.3 Der <myJAM/>-Daemon

Um die Informationen über laufende und wartende Jobs zu sammeln, läuft auf jedem Batchsystem-Server ein <myJAM/>-Daemon-Prozess.

Das Core-Modul des <myJAM/>-Daemon besteht aus einer Schleife über alle (laufenden, wartenden oder angehaltenen) Jobs des Batchsystems. Für laufende Jobs werden vom Batchsystem der aktuell genutzte Arbeitsspeicher, die Walltime und die CPU-Auslastung abgefragt. Zusätzlich ermittelt der Daemon die laufende Anwendung durch Interaktion mit dem Betriebssystem (siehe nächster Abschnitt). Alle gesammelten Informationen werden vom Daemon in die Datenbank geschrieben.

In seltenen Fällen stirbt ein Job, ohne dass das Batchsystem das Epilogskript ausführt, so dass der Job noch in der <myJAM/>-Datenbank als laufender Job geführt wird. Um zu vermeiden, dass sich solche Zombie-Jobs ansammeln, wird in regelmäßigen Abständen eine Garbage-Collection innerhalb des Daemons ausgeführt.

4.4 Anwendungsklassifikation (Software-Accounting) im <myJAM/>-Daemon

<myJAM/> besitzt als ein herausragendes Feature die Fähigkeit, die gerade auf dem Cluster laufenden Anwendungen zu erkennen (*Software-Accounting*). Als Anwendung verstehen wir in diesem Zusammenhang eine ausführbare Datei (Binary), ein Skript oder eine zusammengehörende Sammlung von Binaries und/oder Skripten, die zusammen ein in sich geschlossenes Programm-Paket ergeben.

Die Information, was genau gerade auf welchem Knoten läuft, gehört zu einem umfassenden Monitoring dazu. Die historische Analyse dieser Informationen erlaubt es, Anschaffungen an den Nutzerkreis anzupassen oder gezielt Schulungen für die „Hauptnutzergemeinde“ gezielt anzubieten.

Doch genau hier tun sich die meisten etablierten Tools sehr schwer: Batchsysteme geben hierbei meist nur einen Surrogatschlüssel an (z.B. eine fortlaufende Job-Nummer). Über diesen Surrogatschlüssel sind Zugriffe auf Detailinformationen dieses Jobs möglich, worunter sich auch ein vom User frei wählbarer Jobname befinden kann. Um jedoch zuverlässig anzeigen zu können, welche Anwendung zu diesem Job gehört, wäre man auf die Unterstützung der User angewiesen, die für jeden Job angeben müssen, um was für eine Anwendung es sich handelt. Das ist für die User lästig und insgesamt fehleranfällig. Eine automatische, erweiterbare Erkennung der Anwendung gab es unseres Wissens bisher nicht. <myJAM/> soll diesen Mangel beheben.

Das Betriebssystem kennt den Dateinamen des gerade ausgeführten Binarys oder Skripts. Doch das alleine reicht nicht, da viele Nutzer dazu neigen, jedes ihrer selbst entwickelten Binaries „a.out“ zu nennen, obwohl es sicht *de facto* um völlig unterschiedliche Anwendungen handelt. Daher bietet sich zur eindeutigen Erkennung für Binaries und Skripte eine Hash-Funktion, wie z. B. der Message-Digest-Algorithmus 5 (MD5) von Rivest [Riv92] an. Mit dem MD5-Wert als Schlüssel kann dann aus einer Datenbank abgefragt werden, um welche Anwendung es sich tatsächlich handelt.

Die Hashes der gängigen Binaries und Skripte werden vom Administrator des Clusters in die Datenbank eingepflegt. Unbekannte Hashes (und damit bisher noch nicht erfasste Binaries oder Skripte) werden vorläufig unter ihrem vollen Pfad abgelegt. Der Administrator kann nachträglich die Schlüssel zu bereits bestehenden oder einer neuen Anwendungen hinzufügen. Das Konzept lebt und stirbt damit, wie einfach und komfortabel diese Datenbank aktuell gehalten werden kann. Deshalb haben wir versucht ein durchgängiges Konzept umzusetzen, bei dem man an jeder Stelle des Frontends, an der man mit einem Binary (oder Skript) in Kontakt kommt, durch einen einfachen Klick, den entsprechenden Hash einer Anwendung zuschlagen kann.

4.5 Das <myJAM/>-Web-Frontend

Das <myJAM/>-Web-Frontend ist eine hoch-interaktive Web-Applikation, die für User und Administratoren gleichermaßen das zentrale Interface für das Monitoring von Jobs, Warteschlangen oder des ganzen Clusters, für die Verwaltung von Usern und Projekten, so-

Abbildung 2: Zuordnung von Binaries zu Applikationen für das Software-Accounting in <myJAM/>

wie zur Visualisierung aktueller und historischer Analysen und Statistiken. Das Frontend nutzt neben W3C-konformem XHTML [Pem02] auch objektorientiertes PHP5 [PHP], JavaScript / AJAX und OpenFlashChart [Gla].

4.5.1 Schichtenmodell

Zur Reduzierung von Abhängigkeiten und um die Komplexität möglichst gering zu halten, wurde beim Softwaredesign des <myJAM/>-Web-Frontends ein strenges Schichtenmodell zugrunde gelegt (Abbildung 3).

Backend ist der MySQL-Server. Für den Zugriff auf diesen Server per TCP/IP zeichnet eine eigene Klasse verantwortlich. Sie nimmt MySQL-Anfragen von den Klassen der Schicht darunter entgegen und liefert die Ergebnisse fertig aufbereitet als assoziatives Array zurück. Um Script-Injection-Angriffe abzuwehren, werden alle Sonderzeichen in ihre entsprechenden HTML-Tags konvertiert. Außerdem beinhaltet die Klasse Methoden zur Überprüfung von Usereingaben auf SQL-Injections.

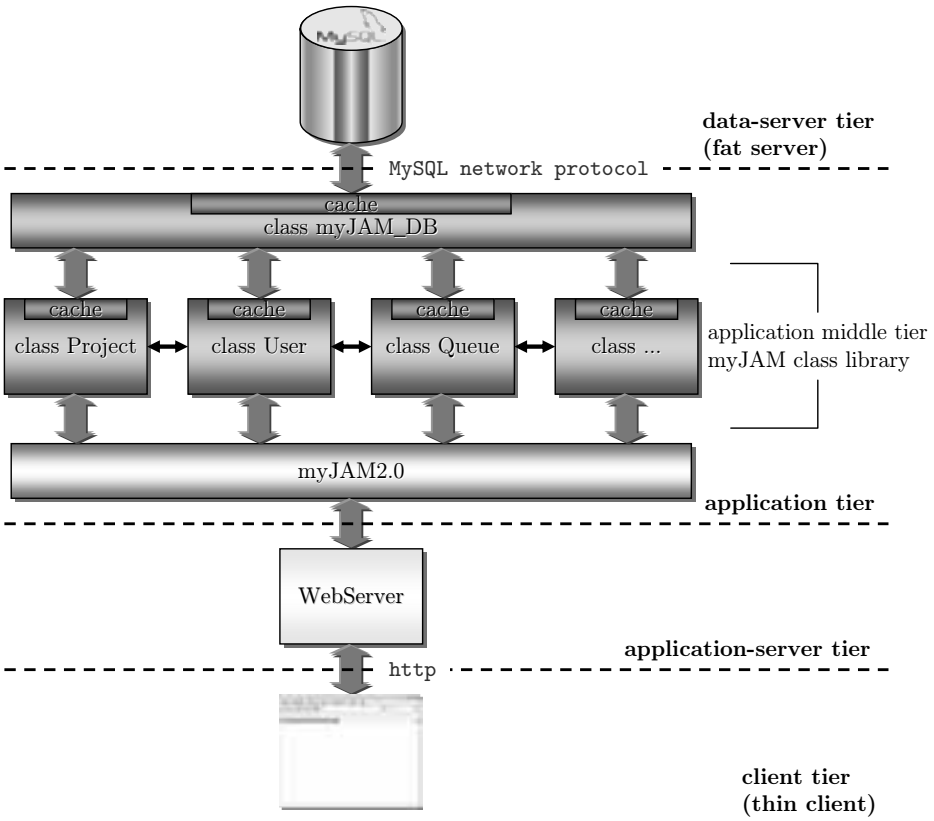

Abbildung 3: Schichtenmodell von <myJAM/>.

4.5.2 Object-Relational Mapping

Die Daten aus den MySQL-Tabellen werden durch *object-relational mapping* (ORM, Objekt-Relationale Abbildung) als PHP5-Objekte von der <myJAM/>-Klassenbibliothek abgebildet. Jedes Objekt eines Typs entspricht dabei einer Zeile derjenigen Tabelle, die diesen Typ repräsentiert. Fremdschlüssel-Primärschlüssel-Beziehungen werden durch Referenzen auf passende Objekte realisiert.

Für die Attribute der Objekte existieren passende *Setters* und *Getters*, in denen direkt erste Sicherheits- und Plausibilitätsüberprüfungen stattfinden. Es können auch komplexe Meta-Attribute, die aus mehreren Einträgen unterschiedlicher MySQL-Tabellen abgeleitet werden, abgefragt werden, z. B. die Anzahl gerade laufender oder wartender Jobs.

4.5.3 Design der Web-Applikation

Ein intuitiv bedienbares und konsistentes Design entscheidet maßgeblich über die Benutzbarkeit einer Anwendung. Das gilt umso mehr bei Web-Applikationen, von denen seitens der Anwender mittlerweile erwartet wird, dass sie selbsterklärend und robust gegen Trial-And-Error-Bedienung sind.

Ein wesentliches Designmerkmal des <myJAM/>-Frontends ist die gegenseitige Verlinkung von Bereichen untereinander (Abbildung 4). Dieses Konzept ermöglicht es, Bereiche und Funktionen auf mehreren Wegen innerhalb von <myJAM/> zu erreichen. User, Projekte, Anwendungen, Jobs, etc. können in jedem Kontext angeklickt werden und führen direkt zu der entsprechenden Detailansicht auf Basis der Methoden dieses Objekts.

Viele Listen- oder Diagrammansichten hängen von Parametern oder Schlüsselwörtern ab. An vielen Stellen wurden AJAX-Methoden implementiert, um diese Ansichten aktualisieren zu können, wenn es eine Useraktion erforderlich macht. So wird die Liste der gelaufenen Anwendungen bereits während der Eingabe des Suchbegriffes aktualisiert.

Die grafische Darstellung von Daten aus der <myJAM/>-Datenbank erfolgt mit Hilfe der Open-Source-Bibliothek „OpenFlashChart“ [Gla]. Diese Bibliothek stellt für verschiedene Diagramm-Typen (wie Linen-, Balken- oder Tortendiagramme) PHP-Objekte zur Verfügung und stellt die Diagramme dann als Flash-Content dar.

4.5.4 Sicherheit

Einer der wichtigsten Grundsätze für Web-Applikationen lautet: Traue nie einer Benutzereingabe. Aus diesem Grunde werden alle Benutzereingabe in <myJAM/> bereits auf Clientseite per JavaScript geprüft – insbesondere auf Script- oder SQL-Injections. Passwörter werden auch bereits auf Clientseite verschlüsselt und dann nur noch das verschlüsselte Ergebnis per POST zum Server geschickt. Trotz der in [WFLY04, WY05] aufgezeigten Kollision der MD5-Funktion erachten wir dieses Verfahren als sicher.

5 Zusammenfassung

Mit <myJAM/> wurde ein leicht und intuitiv bedienbares Frontend für Accounting und Monitoring auf der Basis bereits am Markt etablierter Batchsysteme entwickelt. <myJAM/> wurde in einer heterogenen Cluster-Umgebung entwickelt und erprobt. Durch ein klares Systemdesign und wohldefinierte Schnittstellen können zusätzliche und künftige Batchsysteme sehr leicht unterstützt werden.

<myJAM/> zeigt den aktuellen Status des Clusters in Form von Ressourcen-Auslastung, genutzten Anwendungen – hierfür wurde ein innovatives Software-Accounting implementiert – und in einer Übersicht der Auslastung der verschiedenen Warteschlangen der Batchsysteme. Auch Analysen beliebiger vergangener Zeiträume können durchgeführt werden – jeweils als Verlauf pro Monat oder kumulativ. Durch eigene Kostenmodelle können die

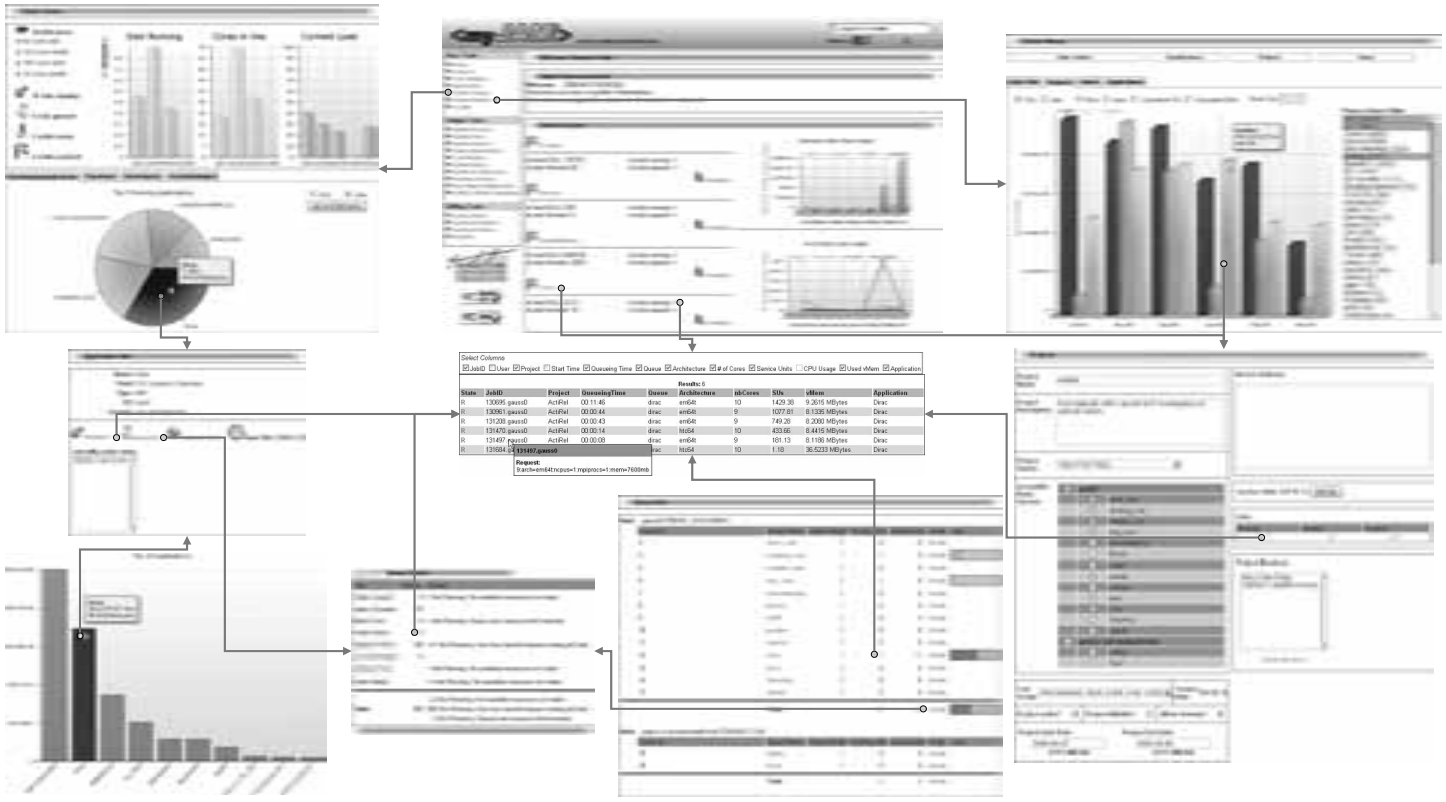

Abbildung 4: Die <myJAM/>-Weboberfläche

von den Projekten genutzten CPU-Stunden monetär abgerechnet oder in Beziehung zu einer Kontingentierung gesetzt werden.

<myJAM/> bietet wertvolle Accounting- und Monitoring-Fähigkeiten zur Unterstützung der Systemadministration, des Nutzersupports und des Berichtswesens. Geplante Erweiterungen sind eine Node-basierte Sicht sowie ein Hochverfügbarkeitsmodul. Darüber hinaus sind weitere Test- und Referenzinstallationen auf großen Clustern geplant, insbesondere in Kooperation mit dem Jülich Supercomputing Centre (JSC).

Danksagung

Die Autoren danken der Firma Bull GmbH für die Projektfinanzierung und die konstruktive Zusammenarbeit sowie dem HPC-Team am ZIM für die Nutzung und betriebliche Unterstützung des Bull-HPC-Clusters zur Entwicklung und Erprobung von <myJAM/>.

References

[Apa] *Apache – HTTP Server Project*. http://httpd.apache.org.

[Gla] John Glazebrook. *The Open Flash Chart project*. http://teethgrinder.co.uk/open-flash-chart.

[MYP] *MyPBS*. http://my-pbs.sourceforge.net.

[PBS07] Altair Engineering, Inc. *PBS Professional 9.0 Administrator's Guide*, 2007.

[Pem02] Steven Pemberton. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). W3C recommendation, W3C, August 2002. http://www.w3.org/TR/2002/REC-xhtml1-20020801.

[PHP] *PHP – A Hypertext Preprocessor*. http://www.php.net.

[Riv92] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Internet Engineering Task Force, April 1992.

[SQL] *MySQL – The world's most popular open source database*. http://www.mysql.com.

[TRQ] *TORQUE Ressource Manager*. http://www.clusterresources.com/products/torque.

[WFLY04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, 2004. http://eprint.iacr.org/.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *EUROCRYPT*, pages 19–35, 2005.