GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

WissGrid

Spezifikation

Arbeitspaket 3: Langzeitarchivierung von Forschungsdaten

WissGrid Dienste Framework[1]

Autoren	Arbeitspaket 3: Langzeitarchivierung von Forschungsdaten
Editoren	Timo Gnadt, Jörg-Holger Panzer, Jens Ludwig, Angelika Reiser, Andreas Aschenbrenner
Datum	16. April 2012
Dokument Status	final
Dokument Version	1.0.0

[1]Diese Arbeit wurde vom WissGrid Projekt erstellt. Dieses Projekt wird gefördert vom Bundesministerium für Bildung und Forschung (BMBF).

Zusammenfassung

Dieses Dokument beschreibt die Anforderungen an das WissGrid Dienste Framework (WDF) zur Kommunikation zwischen Diensten und Repositories, sowie dessen prototypische Implementierung. Neben den Aufgaben und der generellen Architektur des WDF werden die zunächst Themen Scheduling, Allokation von Daten und Rechenleistung behandelt. Anhand der Ergebnisse von Recherchen und Erfahrungen aus anderen Projekten werden hierbei geeignete Komponenten sowie die darüberhinausgehenden notwendigen Anpassungen beschrieben. Im Weiteren werden durchgeführte Testreihen zum Scheduling dargestellt und ausgewertet und die realisierte prototypische Implementierung erläutert. Schließlich wird am Beispiel eines Anwendungsfalles im Projekt TextGrid eine mögliche Anwendung des WDF skizziert.

Inhaltsverzeichnis

1 Einleitung

Das WissGrid Dienste Framework (WDF) dient zur Ausführung und Koordinierung der LZA-Dienste sowie der Kommunikation zwischen Diensten und Repository. Obwohl das WDF eine zentrale Rolle in der Einbettung von LZA-Diensten in ein Repository hat, ist es eine von dem Repository unabhängige Komponente. Als solches ist das WDF generisch und auch zur Kommunikation mit mehreren, unterschiedlichen Repositorien ausgelegt.

Eine grundlegende Unterscheidung in der Koordination der Daten und der LZA-Dienste liegt in der Frage: *Daten zu den Diensten, oder Dienste zu den Daten?* Eine Entscheidung hierüber hängt einerseits ab von Performance-Fragen (Wie groß sind die Daten? Wie rechenintensiv ist die Ausführung des Dienstes auf den spezifischen Datenformaten?), andererseits aber auch von Sicherheitsfragen (Dürfen externe/fremde Dienste innerhalb eines vertrauenswürdigen Langzeitarchivs ausgeführt werden?) und letztlich der Architektur des Repositories (Lässt die Softwarearchitektur und die (verteilte) Hardware die Ausführung von LZA-Diensten zu?). Daher muss jede Community diese Entscheidung für sich treffen. Im Rahmen des WissGrid-Projektes wird aufgrund von Sicherheitsbedenken, und mit dem Ziel einer möglichst generischen Lösung nur das Muster „Daten zu den Diensten“ verfolgt. Partnerprojekte (z.B. AstroGrid) experimentieren aber auch mit Community-spezifischen Ansätzen für „Dienste zu den Daten“, und die dort gesammelten Erfahrungen werden perspektivisch in WissGrid einfließen.

Dieses Dokument stellt eine zweite Fortführung des Deliverables 3.4.2 zur LZA-Spezifikation [14] dar. Die folgenden Abschnitte beschreiben die Aufgaben (Abschnitt 2) und die Architektur (Abschnitt 3) des WDF, sowie Anforderungen und mögliche Lösungen für die erforderliche Scheduling-Komponente (Abschnitt 4). In Abschnitt 5 werden die mit einem ausgewählten Scheduler durchgeführten Tests kurz vorgestellt und deren Ergebnisse ausgewertet. Abschnitt 6 erläutert die realisierte prototypische Implementierung des WDF als Scheduler mit Anbindung an das WissGrid-Repository. Abschließend findet sich in Abschnitt 7 eine Spezifikation für den Anwendungsfall TextGrid-Masseningest.

2 Aufgaben des WDF

Das WDF koordiniert die Kommunikation zwischen einem Repository und vorhandenen LZA-Diensten. Dieser Abschnitt konzentriert sich auf LZA-Dienste, die als Grid-Dienste genutzt werden. Die Nutzung von Web Services für LZA wie dem Planets Interoperability Framework [6] ist auf einer höheren Applikationsschicht anzusiedeln, und deren Integration in das WDF wird derzeit nicht avisiert.

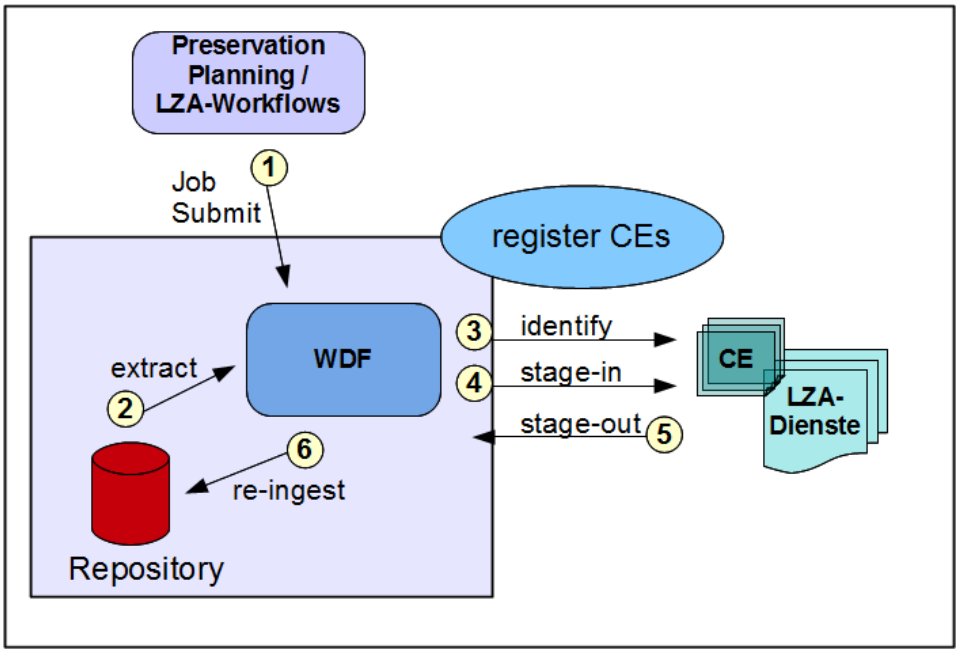


Abbildung 1: Interaktion zwischen Diensten und Repository durch WDF

In Abbildung 1 ist die Kommunikation zur Ausführung einer typischen LZA-Aufgabe schematisch dargestellt. Die Hauptkomponenten sind hierbei

- der Auslöser einer LZA-Aufgabe, wobei hier zwischen dem organisatorischen, durch Menschen gesteuerten Planungsvorgang und Beschluss (z.B. die Preservation Planning Einheit eines Forschungsarchivs) einerseits, und der technischen Durchführung (z.B. durch Trigger) andererseits zu unterscheiden ist
- das Repository, in dem die Daten vorliegen
- der WDF-Dienst, der meist eng mit einem Repository verknüpft ist und auch in einer gemeinsamen Sicherheitsdomäne angesiedelt ist
- die LZA-Dienste, die auf Compute Entities (CEs) installiert sind und über Grid-Mechanismen angesprochen werden können

Die Beschreibungen von LZA-Diensten und der zugehörigen CEs werden hierbei explizit beim WDF zur Nutzung registriert.

Der Ablauf einer LZA-Aufgabe wird über das WDF koordiniert und beinhaltet folgende Schritte:

1. **Submit:** Annahme der LZA-Aufgabe, z.B.:
   - Konvertiere alle TIFF Daten, die älter als 2 Jahre sind, in ein JPEG 2000 Format
   - Extrahiere aus eingehenden Bilddateien alle Metadaten, welche Erstellung und Bearbeitung betreffen
2. **Extract:** Filterung der gefragten Daten aus dem Repository *oder* Annahme von Daten aus einem **Ingest**
3. **Identify:** Auswahl einer verfügbaren CE, auf der der gefragte Dienst installiert ist
4. **Stage-In:** Transfer der Daten (im Batch) auf das CE
5. **Stage-Out:** Annahme der konvertierten Daten, nach Ablauf des Jobs; ggf. Zwischenspeicherung der Daten
6. **Re-Ingest:** Rückführung in das Repository, gegebenenfalls unter neuerlicher Vergabe von Metadaten und Erstellung einer neuen Objekt-Version

3 Architektur

In diesem Abschnitt wird die Architektur des WissGrid Dienste Frameworks vorgestellt, in welcher die möglichen bzw. erforderlichen Schnittstellen des WDF zum Repository, zum Nutzer und zu den Diensten im Grid spezifiziert werden.

Der Einsatz des WDF ist an mehreren unterschiedlichen Stellen denkbar:

- aus der Administration des Repositories (Archival Information Update, OAIS) heraus, wenn eine Routine LZA-Aufgabe im laufenden Betrieb erfüllt werden muss, z.B. wenn alle Objekte in einem veralteten Format in ein aktuelles Format migriert werden
- beim Einbringen neuer Objekte in das Repository („Ingest“), z.B. zur Validierung von Dateiformaten
- beim Zugriff auf Repository-Inhalte („Access“), z.B. bei einer Migration-on-Access LZA-Strategie, bei der die Inhalte als Originale oder in einem Standardformat im Repository vorgehalten werden, und nur beim Zugriff in ein vom Nutzer gewünschtes Format überführt werden
- für Nutzer-gesteuerte Workflows, in denen LZA-Dienste benötigt werden

Um diese generische Einsetzbarkeit zu erreichen, ist das WDF als unabhängige Komponente konzipiert, die aber physisch „nahe“ an einem Host-Repository installiert wird

(a) um maximale Datenübertragungsraten für die oft umfangreichen zu transportierenden Datenmengen zu erreichen, und auch

(b) um ein Repository und ein zugehöriges WDF in einer gemeinsamen Sicherheitsdomäne zu behalten und damit Rechtefragen zu vereinfachen.

Idealerweise sollte somit für jedes Repository ein WDF installiert werden.

Abbildung 2 zeigt die schematische Einbettung des WDF in den Kontext zwischen virtueller Forschungsumgebung (Virtual Research Environment, VRE), Repository (wie beschrieben in [13]) und dem Grid. Die Funktionalitäten der in Abbildung 2 gezeigten internen Komponenten des WDF schlüsseln sich wie folgt auf:

1. **Service Trigger:** Bietet Schnittstellen zum Client (VRE), nimmt Anfragen entgegen und extrahiert bzw. sammelt benötigte Informationen:
   1.1 Welche Operation wird angefordert? Weiterleitung an Repository-Schnittstelle
   1.2 Welche Dienste müssen zur Abarbeitung ausgeführt werden? Anfrage an Service Catalogue
   1.3 Wieviele Dateien in welcher Größe sind von der Dienstausführung betroffen? Entscheidungsfindung zu verteilter Ausführung über/für Meta-Scheduler
2. **Service Catalogue:** Verwaltet Zusammenhang (Mapping) von Vorgängen (s. Service Trigger) und auszuführenden Diensten
3. **Service Registry:** Verwaltet aktuelle Verteilung von Dienstinstallationen und Compute Entities

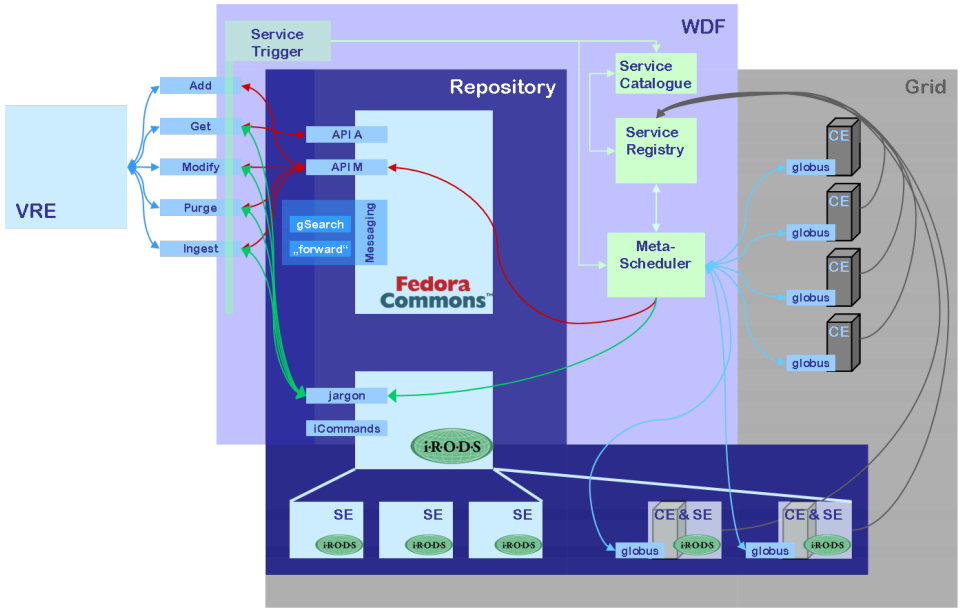


Abbildung 2: Einbindung des WDF - Schematische Übersicht

4. **Meta-Scheduler:** Organisiert verteilte Ausführung von Diensten, Datentransport zu den CNs, Ergebnistransport zum Repository Schnittstellen

Die benötigten Schnittstellen wurden, wie auch die Registrierung der verfügbaren Compute Entities (CEs) und deren angebotenen LZA-Dienste, im Rahmen von WissGrid lediglich hart-kodiert. Im Folgenden werden diese Schnittstellen benannt und ihre generische Funktionalität kurz beschrieben.

3.1 Schnittstellen

Das WDF benötigt Schnittstellen zur VRE, zum Grid und zum Repository.

1. **Zum Client (VRE):** Eingangsschnittstellen für alle möglichen generellen Vorgänge:

   Ingest: Der Client will eine oder mehrere Dateien im Repository speichern

   Add: Der Client will Metadaten zu einer oder mehreren Dateien im Repository hinzufügen

   Modify: Der Client will eine oder mehrere Dateien im Repository verändern (inkl. Versionierung)

   Purge: Der Client will eine oder mehrere Dateien aus dem Repository entfernen

   Get: Der Client will eine oder mehrere Dateien aus dem Repository laden

2. **Zum Grid (Compute Entities):** Ein- und Ausgangsschnittstellen

   Register: Compute Entities müssen sich und die auf Ihnen ausführbaren Dienste beim WDF registrieren (Service Registry)

   Scheduling: Compute Entities werden für die Dienstausführung ausgewählt. Beinhaltet Datenübertragung und Dienstmonitoring (wie z.B. realisiert durch Globus/GridWay)

3. **Zum Repository:**

Relay: Durchreichen der Vorgänge unter 1)

3.1 Ingest, Modify, Purge an Fedora API-M bzw. iRODS jargon

3.2 Add an Fedora API-M

3.3 Get an Fedora API-A bzw. iRODS jargon

Update: Hinzufügen der Resultate (Metadaten) aus Dienstausführung

3.1 Add an Fedora API-M

3.2 Modify an Fedora API-M bzw. iRODS jargon

4 Scheduling-Aufgaben im WDF

Eine Hauptaufgabe des WDF liegt in der Verwaltung und Steuerung von im Grid verfügbaren Diensten. Um die Ausführung und Abarbeitung dieser Dienste effizient zu gestalten, ist ein sogenannter Scheduler erforderlich, welcher die verfügbaren und geeigneten Computing Elements (CEs) identifiziert und sowohl den ggf. erforderlichen Datenfluss von den Storage Elements (SEs) zu den CEs als auch die Dienstausführung steuert und überwacht. In diesem Abschnitt werden die konkreten Anforderungen an einen solchen Scheduler benannt und mit den Spezifikationen von existierenden Schedulern abgeglichen.

4.1 Anforderungen an Scheduler

Die vom Scheduler konkret zu verrichtenden Aufgaben hängen zum Teil von den jeweils auszuführenden Diensten ab. Je nach Dienst muss ebenfalls entschieden werden, ob ein Scheduling überhaupt notwendig und sinnvoll ist. So sind zum Beispiel im Anwendungsfall des TextGrid-Masseningest die Parallelisierung der Dienstausführung sowie die Berücksichtigung des Datentransferaufwandes („data awareness“) entscheidende Kriterien, welche den Einsatz eines Schedulers erforderlich machen. In anderen Fällen (z.B. Ingest eines einzelnen Objekts) kann die Dienstausführung jedoch ggf. ohne Scheduling direkt auf dem Rechner des Ziel-Repositories erfolgen. Innerhalb des WDF muss somit eine Instanz in jedem Fall darüber entscheiden, ob und in welchem Umfang der Scheduler eingesetzt wird. Eine zu klärende Frage besteht somit darin, ob diese Entscheidung vom verwendeten Scheduler selbst getroffen werden kann, oder ob hierfür eine Erweiterung des Schedulers bzw. eine eigenständige Komponente im WDF erforderlich ist.

Eine andere Frage betrifft die vom WDF zu speichernden Informationen über die CEs im Grid. Eine Anforderung für die sinnvolle Verteilung der Dienstausführungen im Grid ist zunächst, dass sämtliche verfügbaren CEs beim WDF registriert werden, einschließlich der bei ihnen installierten Dienste. Zu erwägen ist weiterhin, ob darüber hinaus auch durchschnittliche oder garantierte Transferraten von und zu den CEs gespeichert werden können und sollen. Diese Informationen könnten dann im WDF direkt mit den Informationen über die für einen Prozess (Ingest, Access, etc.) notwendigen Dienste sowie den Anforderungen an selbige verknüpft und dem Scheduler zur Verfügung gestellt werden. Als einzige vom WDF bei den CEs direkt zu erfragende Information verbliebe dann die aktuelle Auslastung der CEs hinsichtlich Speicher und Rechenkapazität.

Um eine möglichst generische Struktur des WDF zu gewährleisten, welche mit verschiedenen Repository-Systemen und Grid-Ressourcen zusammenarbeiten kann, sollten bei der Wahl eines Schedulers alle möglichen Fälle berücksichtigt werden.

Unabhängig von der konkreten Implementierung des WDF sollte der eingesetzte Scheduler folgende Kernaufgaben erfüllen können:

- **Stage-In:** Verteilung und Transfer der Daten auf die CEs
- Aufruf der Dienste auf den CEs
- **Stage-Out:** Annahme der Dienst-Ergebnisse und Weiterleitung an Nachbearbeitung

- Überwachung der Transfers und Dienst-Ergebnisse auf Fehler, sowie Behandlung dieser Fehler (z.B. Re-Scheduling)

In Abhängigkeit vom eingesetzten Scheduler und dessen Anpassbarkeit sowie vom Leistungsumfang der übrigen Repository-Komponenten ist zu entscheiden, ob einige der folgenden Aufgaben ebenfalls vom Scheduler übernommen werden können:

- **Submit:** Annahme der Daten vom Producer (s. OAIS-Modell)
- **Extract:** Extrahieren der erforderlichen Daten aus dem Repository
- Entscheidung über Einsatz und Umfang von Scheduling
- **Identify:** Ermittlung der geeigneten CEs einschließlich Berechnung und Abwägung des Transfer- und Rechenaufwands
- Aufteilung der Daten in für Transfer und Dienstausführung geeignete Pakete (bei Masseningest oder -access)

Sinnvollerweise nicht direkt vom Scheduler, sondern von anderen Komponenten zu übernehmende Tasks sind hingegen:

- Ermittlung der für den jeweiligen Prozess auszuführenden Dienste sowie der Anforderungen an die CEs hinsichtlich Transferrate und Rechen- sowie Speicherkapazität
- Aufruf der Nachbearbeitungs-Prozesse (PID-Vergabe, Suchindex-Erstellung etc.)
- **(Re-)Ingest:** Speicherung bzw. Aktualisierung der Daten im Repository

4.2 Existierende Scheduler

In diesem Abschnitt werden einige gebräuchliche Scheduler anhand ihrer eigenen Spezifikationen kurz vorgestellt und im Hinblick auf die in Abschnitt 4.1 aufgeführten Anforderungen hinsichtlich ihrer Eignung für den Einsatz im WDF bewertet.

GridWay

Dieser Scheduler stammt aus der Globus Community [2] und integriert sich folglich nahtlos in das weit verbreitete Globus Toolkit. GridWay [10] wird unter der Bezeichnung „Workload Manager“ zur Verwaltung von Diensten und Ressourcen im Grid geführt. Die hervorstechendsten Merkmale dieses Schedulers sind die umfangreiche Fehlerbehandlung, die Fähigkeit zu dynamischem Scheduling inklusive Migration sowie die Unterstützung von Scheduling Policies und voneinander abhängigen Diensten. Eine Berücksichtigung von data awareness ist hier noch nicht implementiert, allerdings existiert ein erster Prototyp hierzu [16], welcher jedoch noch nicht als Software veröffentlicht wurde und somit in WissGrid nicht für Tests zur Verfügung stand.

CSF

Das Community Scheduler Framework (CSF) [1] ist ebenfalls ein Produkt der Globus Community und wurde bereits in der Globus Toolkit Version 3.0 integriert. Es handelt sich hierbei um eine Open-Source Implementierung von verschiedenen Grid-Services, welche zusammen die Funktionalität eines Grid-Meta-Schedulers umfassen. Für die konkrete Anwendung im Grid kann das CSF als Baukasten für sogenannte Community Scheduler genutzt werden, welche dann auf die jeweilige Anwendung und Grid-Struktur der entsprechenden Virtuellen Organisation zugeschnitten sind. Das CSF bietet eine Abstraktionsebene für Konzepte wie „Job“, „Ressourcen-Reservierung“ oder „Scheduling“ und forciert die Einhaltung von Standards wie des OGSI-Agreements [8]. Konkrete Mechanismen zur Berücksichtigung des Datentransferaufwands sind hier jedoch nicht erkennbar.

GWES

Der Generic Workflow Execution Service (GWES) [9] wurde vom Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik (FIRST) entwickelt. Er verwendet eine eigens entwickelte Beschreibungssprache (GWorkflowDL), welche auf High-Level Petri-Netzen basiert und eine umfassende Beschreibung und Modellierung von Workflows erlaubt. GWES wird zur Beschreibung von Workflows bereits in mehreren Grid-Projekten angewendet, darunter auch TextGrid [18]. Die Unterstützung von data allocation scheint sich der Spezifikation nach jedoch auf voneinander abhängige Dienste innerhalb eines Workflows zu beschränken und keine grundsätzlichen Erwägungen bei der Auswahl der dienst-ausführenden Ressourcen in Abhängigkeit vom Speicherort der Daten zu beinhalten.

WSS

Der Workflow Scheduling Service (WSS) ist eine Scheduler-Entwicklung des Projektes C3Grid [4]. Er wurde für die direkte Interaktion zwischen Benutzer, Repository und Diensten entwickelt und wird im Rahmen von C3Grid von Benutzern direkt über ein Portal angesprochen. Seine Aufgaben liegen in der Koordination von Task-Ausführungen im Grid sowie im Planen von Datenzugriffen, wobei er eng mit dem C3Grid Data Management System (DMS) zusammenarbeitet. In diesem Zusammenhang wird auch das Problem der Datenallokation betrachtet.
Um die Verwaltung von gegenseitig abhängigen Diensten zu unterstützen, wurde hierbei eine eigene, auf der standardisierten Job Submission Description Language (JSDL) aufbauende Sprache zur Dienstebeschreibung entwickelt, die Workflow Specification Language (WSL). Die Überwachung von Dienstausführungen wird jedoch laut Spezifikation nicht vom Scheduler WSS selbst geleistet, sondern vom C3Grid-Portal koordiniert.

Die Recherchen im Rahmen von WissGrid zum Scheduling-Ansatz des C3Grid unterstützen die Einschätzung, dass eine direkte Nutzung des WSS in einem externen Kontext derzeit kaum zu realisieren ist. Dies ist unter anderem auch durch die aktuelle Umstellung des gesamten Systems aus C3Grid in dessen Nachfolgeprojekt C3-INAD [3] bedingt, welches die Unterstützung der GLOBUS-Middleware voraussichtlich aufgibt und eigene Lösungen entwickelt.

KOALA

Beim Co-allocating Grid Scheduler KOALA [5], einer Entwicklung der TU Delft im Rahmen des VL-e-Projekts [15], steht die Integration von Daten- und Prozessor-Zuweisungen für daten-intensive Grid-Jobs im Vordergrund. Der Scheduler identifiziert dabei zunächst geeignete Ressourcen im Grid, transferiert dann die Daten zu diesen Ressourcen und bemüht sich erst anschließend um die Reservierung von Rechenkapazitäten. Hinsichtlich data-awareness kann KOALA verschiedene Co-Allocation policies verfolgen, so z.B. Load-aware (WF), Input-file-location-aware (CF) oder Communication-aware (CM/FCM/CA) [17]. Die Entwicklung von KOALA scheint jedoch seit dem Projektende 2009 nicht weitergeführt worden zu sein — auch lässt sich keine Dokumentation zur Verwendung dieses Schedulers in anderen Umgebungen finden.

WisNetGrid

Im Projekt WisNetGrid [20] wird an einer generischen Lösung für die Beschreibung von Diensten und Workflows gearbeitet. Ein erstes Ergebnis hierbei ist eine eigene Dienstbeschreibungssprache suprimePDL [19], welche auf dem π-Kalkül basiert und verschiedenste Anforderungen zur allgemeinen Modellierung und Beschreibung von Prozessen abdeckt, u.a. auch Dienstekomposition. Eine Architektur oder gar Anwendung zur Erfüllung von Scheduling-Aufgaben ist hier bisher nicht vorhanden. Auch Fragen der data-awareness werden in WisNetGrid bisher nicht angesprochen.

Zusammenfassung

Aus den in diesem Abschnitt beschriebenen Spezifikationen sowie aus Gesprächen mit Partnern aus anderen Projekten (C3Grid, TextGrid, WisNetGrid) wurden im Rahmen von WissGrid folgende Schlüsse gezogen:

- Viele zur Zeit vorhandene Scheduler sind das Ergebnis von akademischen Projekten und als solche hinsichtlich ihrer weiteren Unterstützung nicht gesichert.
- Die konkrete Eignung eines Schedulers bezüglich seiner Anpassungsfähigkeit an verschiedene Grid-Kontexte und -Anforderungen lässt sich nur durch die Implementierung geeigneter Testfälle abschätzen.
- Die Implementierung von data-awareness, also der Berücksichtigung des erforderlichen Datentransferaufwandes bei der Jobverteilung, ist bisher kaum verbreitet.
- Der Fokus in WissGrid sollte zunächst auf der genauen Spezifikation von Anwendungsfällen und deren Anforderungen, sowie der prototypischen Umsetzung dieser Anwendungsfälle mit möglichst weit verbreiteten Schedulern liegen.
- Zur Verwendung in diesen Anwendungsfällen wurde daher eine erste Lösung mit der GLOBUS Middleware und dem Scheduler GridWay umgesetzt und für Tests verwendet. Die durchgeführten Tests sowie die prototypische Umsetzung der Repository-Anbindung werden in den folgenden beiden Abschnitten näher beschrieben.

5 Ergebnisse

Im Rahmen von WissGrid wurden 2 Testreihen mit dem Scheduler GridWay (Version 5.8) durchgeführt, um die Performance der verteilten Jobbearbeitung besser einzuschätzen und eine Empfehlung dafür zu entwickeln, ab welcher Anzahl, Datentransfergröße und Verbindungsgeschwindigkeit eine entfernte Dienstausführung sinnvoll ist. Im Folgenden werden zunächst die beiden Testreihen kurz beschrieben und deren Ergebnisse erläutert und evaluiert. Abschließend werden die insgesamt abzuleitenden Erkenntnisse und deren Auswirkungen auf die Scheduling-Entscheidung skizziert.

5.1 Scheduling-Testreihe zur Job-Verteilung

Bei diesem Test wurden folgende drei Rechner bzw. virtuelle Maschinen verwendet:

Nr.	Name	Globus Version	Betriebssystem	CPU	CPU Cache	RAM
1	theGlobus5	5.0.2	(VM) Ubuntu 10.10	Intel Core2Duo, 2.93 GHz	6MB	1GB
2	ingrid	4.2.1	Suse Linux 10.1	2x Intel Pentium D, 3.4GHz	2MB	8GB
3	astrodata04	4.0.8	Scientific Linux 5.5	2x AMD Opteron 250, 2.4GHz	1MB	8GB

Tabelle 1: Verwendete Compute Entities (CEs) bei den Job-Verteilungs-Tests

Während die Rechner 1 und 2 über eine schnelle Intranet-Verbindung kommunizieren konnten, war Rechner 3 nur über eine Internet-Verbindung angebunden.

Über verschiedene Testläufe wurden folgende Variablen variiert:

1. Die Art des Jobs
   a) „DATE“: Konsolenausgabe des aktuellen Datums
   b) „JHOVE2“: Formatvalidierung einer von 103 Bilddateien mit JHOVE2 [12]
   c) „FITS“: Formatvalidierung einer von 103 Bilddateien mit FITS [7]
2. Der Job-initiierende Rechner
   a) „THE“: Rechner 1
   b) „INGRID“: Rechner 2
3. Die Anzahl der Compute Entities
   a) normal: alle drei oben aufgeführten Rechner
   b) „local“: nur der Job-initiierende Rechner

4. Die Pfadangaben für die Eingabedateien sowie die Job-Ausführungsdatei in den GridWay Job Templates
   a) normal: relativ, d.h. GridWay überträgt die Dateien
   b) „DIST“: absolut, d.h. GridWay benutzt die Dateien lokal auf dem jeweils ausführenden Rechner

Jeder Testlauf bestand in der Abarbeitung von 103 Jobs. Dabei wurden jeweils folgende Werte unabhängig für jeden Rechner gemessen:

1. Die Anzahl der erfolgreich ausgeführten Jobs
2. Die durchschnittliche Transferzeit eines Jobs (GridWay-Messung)
3. Die durchschnittliche Ausführungszeit eines Jobs (GridWay-Messung)

Außerdem wurde jeweils die tatsächliche Gesamtdauer einer Jobsequenz gemessen. Im Folgenden werden nur die Ergebnisse der JHOVE2-Jobausführungen grafischen dargestellt. Die Ergebnisse für die anderen Jobs sowie alle weiteren Ergebnisse finden sich im Anhang C.

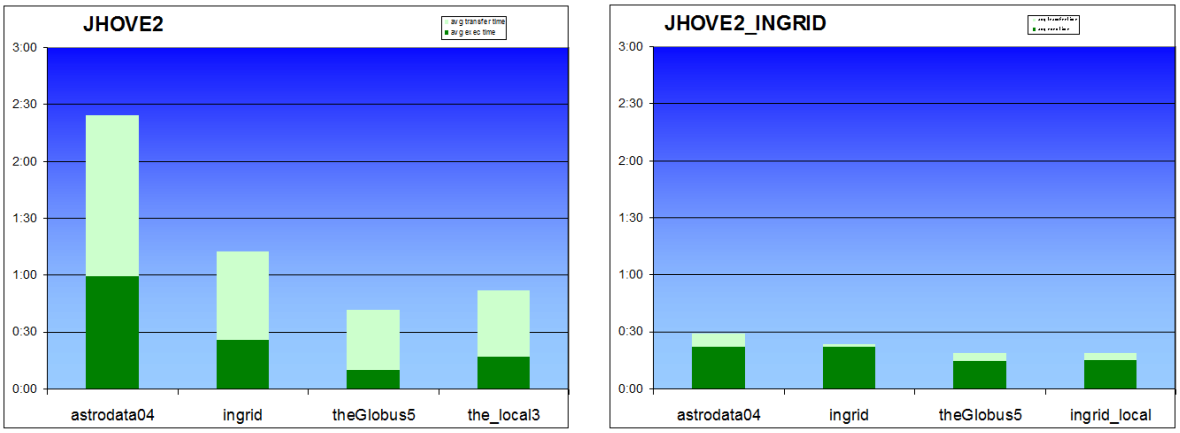
Als wesentliches Problem erwies sich bei der Auswertung der Umstand, dass GridWay bisher nicht über die Möglichkeit zur eigenständigen Datenlokalisation verfügt, also wie schon in Abschnitt 4.2 beschrieben keine data-awareness realisieren kann. Stattdessen werden in jedem Fall Dateien zum entfernten Rechner hin und von ihm zurück übertragen. Auch wenn (wie unter Punkt 4b beschrieben) die zu verarbeitenden Dateien lokal auf dem ausführenden Rechner vorliegen und verarbeitet werden können, überträgt GridWay dennoch die Jobbeschreibungsdatei („Job Template“) sowie diverse kleine Hilfsdateien (sog. „Wrapper-Skripte“). Der Zeitaufwand für den Datentransfer alleine erwies sich bei Dateigrößen von wenigen MB jedoch als vernachlässigbar - der Aufbau der Verbindung und die Kommunikation bzgl. des Job-Status zeigten sich bei diesen Tests als die maßgeblichen Größen.

Dies resultiert insbesondere in (von GridWay gemessenen) relativ langen Job-Ausführungszeiten bei Jobs, welche auf entfernten Compute Entities bearbeitet wurden. Hierbei wurden nämlich teilweise auch die Kommunikationszeiten als Ausführungszeit erfasst, bzw. wurde der Abschluss einer Job-Ausführungsphase von GridWay aufgrund der verzögerten Kommunikation oft erst verspätet registriert.

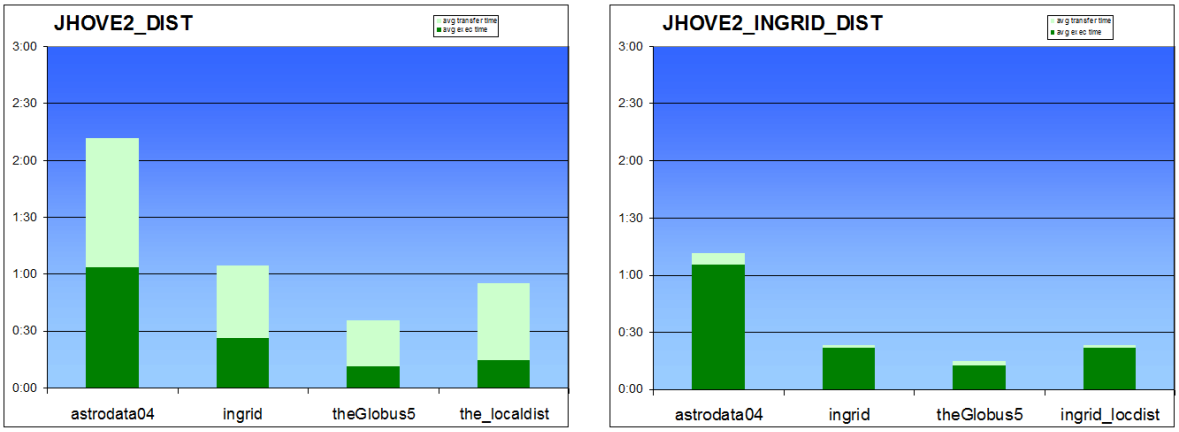
Die Abbildungen 3(a) und 3(b) zeigen die durchschnittlichen Ausführungszeiten für die JHOVE2-Jobs unter verschiedenen Konfigurationen. Bei den in Abbildung 3(a) dargestellten Tests wurden die Eingabe- und Ausführungsdateien mit übertragen („Daten zu den Diensten“), bei den in Abbildung 3(b) abgebildeten Tests wurden die Daten lokal auf den jeweils ausführenden Rechner verwendet.

In den Abbildungen 3(a) und 3(b) zeigen die linken Diagramme jeweils die Ergebnisse mit job-initiierendem Rechner 1, die rechten Diagramme Ergebnisse mit initiierenden Rechner 2. Die einzelnen Säulen bestehen dabei aus der Ausführungszeit (unten, dunkelgrün) und der Transferzeit (oben, hellgrün). Während die ersten drei Säulen jedes Diagramms die durchschnittliche Ausführungszeit auf den drei Rechnern bei verteilter Job-Ausführung abbilden, zeigt die vierte (rechte) Säule die durchschnittliche Ausführungszeit bei nicht-verteilter Ausführung, d.h. Ausführung nur auf dem lokalen Rechner.

Eine Erkenntnis lässt sich aus den in den rechten Spalten dargestellten deutlich kürzeren Ausführungs- wie auch Transferzeiten für den Rechner 3 (astrodata04) ableiten: Jene Ergebnisse wurden mit



(a) mit Transfer von Eingabe- und Ausführungsdatei



(b) ohne Transfer von Eingabe- und Ausführungsdatei

Abbildung 3: Durchschnittliche JHOVE2-Job-Ausführungszeiten in Minuten

dem Job-initiierenden Rechner 2 erzielt, welcher Globus Version 4.2.1 und somit die dort noch unterstützen Webservices verwendet (im Gegensatz zu Globus Version 5 bei Rechner 1, also in den linken Diagrammen). Hier funktionierte die Kommunikation mit Rechner 3 (Globus Version 4.0.8) also deutlich besser, wodurch die Jobs schneller verteilt und insgesamt somit schneller bearbeitet wurden.

Maßgeblicher für die Entscheidung für oder gegen eine verteilte Ausführung sind jedoch die Ergebnisse bezüglich der gesamten benötigten Transferzeit, Rechenzeit und Dauer für eine Jobsequenz. Abbildung 4 zeigt die Ergebnisse exemplarisch für JHOVE2-Job-Ausführungen. Hierbei sind auf der x-Achse verschiedene Konfigurationen aufgeführt. Die Einträge 1,4,5 und 8 (beginnend mit „INGRID“) stehen hierbei für den job-initiierenden Rechner 2, die übrigen für den job-initiierenden Rechner 1. Bei den Einträgen mit der Endung „DIST“ erfolgte während der Jobausführung kein Transfer der Eingabe und Ausführungsdateien, Einträge mit „local“ bzw. „loc“ bezeichnen die nicht-verteilte Ausführung der Jobs.

Zunächst fällt ins Auge, dass die tatsächliche Dauer („Duration“) der Jobsequenzen die akkumulierten Ausführungs- („Execution“) und Transferzeiten („Transfer“) stark unterschreitet, wenn eine verteilte Ausführung gewählt wurde (linken 4 Einträge), jedoch deutlich höher liegt, wenn die Jobs nur auf

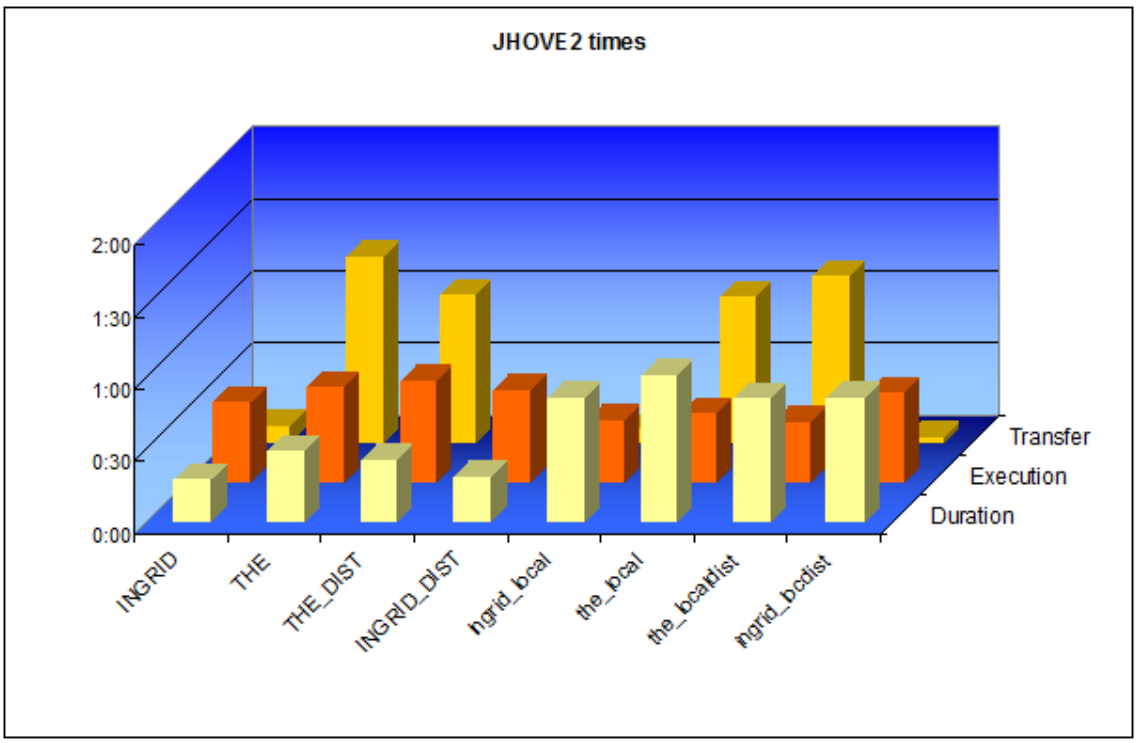


Abbildung 4: Durch GridWay gemessene Zeiten (in Stunden) von Jobsequenzen à 103 JHOVE2-Jobs unter verschiedenen Konfigurationen.

einem Rechner bearbeitet wurden. Insbesondere in Anbetracht der leicht höheren Ausführungszeiten bei der verteilten Ausführung ist dies bemerkenswert. Dieser Performance-Gewinn wird somit allein durch die Parallelisierung der Jobs erreicht und spricht daher bei den verwendeten Dateigrößen (wenige MB) eindeutig für eine verteilte Ausführung, wenn keine weiteren Hindernisse bzgl. der Inanspruchnahme der Rechenzeit auf den entfernten Compute Entities bestehen.

Außerdem wird ersichtlich, dass die Verwendung von lokal vorliegenden Eingabe- und Ausführungsdateien (d.h. eine Vermeidung ihres Transfers, Konfigurationsnamen mit „DIST“) keine wesentlichen Performance-Verbesserungen bringt. In einem Fall zeigt sich sogar eine leichte Verschlechterung, welche aber unter statistischen Gesichtspunkten vernachlässigbar ist.

Eine weitere Besonderheit zeigt sich in relativ hohen Transferzeiten beim job-initiierenden Rechner 1, sowohl bei der verteilten, als auch bei der lokalen Ausführung. Hierfür sind, wie oben bereits erläutert, höchstwahrscheinlich die verschiedenen Globus-Versionen bzw. die Nicht-Unterstützung von Webservices durch Globus5 verantwortlich zu machen.

Die Verteilung von JHOVE2-Jobs über die verschiedenen Compute Entities ist in Abbildung 5 dargestellt. Die gezeigten Konfigurationen entsprechen denen in Abbildung 4. Hier lässt sich eine recht effektive, gleichmäßige Verteilung für den ersten Fall (linke Säule) erkennen, bei dem sowohl die Globus-Unterstützung für Webservice auf dem job-initiierenden Rechner gegeben war als auch die Eingabe- und Ausführungsdateien übertragen wurden. Ansonsten zeigt sich, dass aufgrund des schon genannten Kommunikations-Overheads bei verteilten Ausführungen die meisten Jobs erwartungsgemäß eher auf dem job-initiierenden Rechner ausgeführt werden.

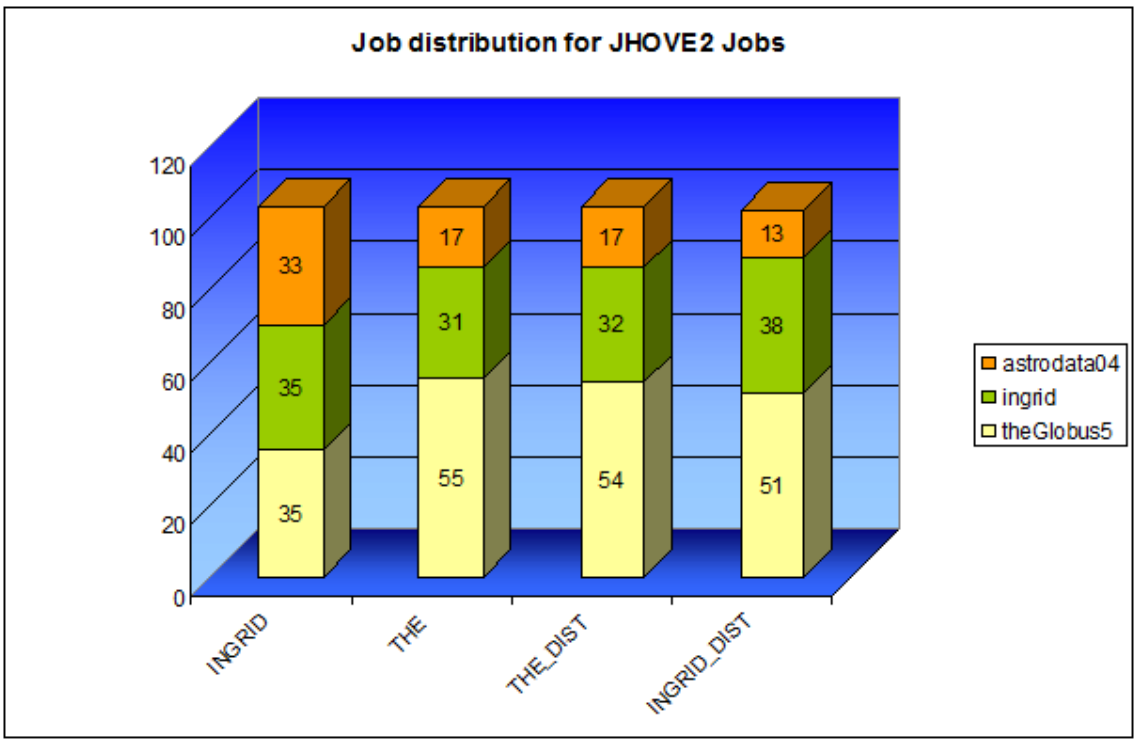


Abbildung 5: Job-Verteilung von 103 JHOVE2-Jobs über verwendete CEs unter verschiedenen Konfigurationen.

5.2 Scheduling-Testreihe zum Datentransfer

Ziel dieser Testreihe war es zu untersuchen, welchen Einfluss die Größe der zu übertragenden Dateien und die Anzahl an Jobs auf die durchschnittliche Gesamtdauer haben. Interessant sind hierbei sich gegenseitig beeinflussende Effekte: Erhöht sich die Jobdauer durch eine größere Transferdatei, so sollte dies den Scheduler zu einer verstärkten Jobverteilung veranlassen. Diese würde jedoch wiederum zu einem erhöhten Transferaufkommen führen, wobei sich irgendwann eine relativ konstante Jobdauer einstellen müsste.

Bei diesen Tests wurden 5 Rechner verwendet, wobei 4 davon eine identische Konfiguration aufweisen:

Nr.	Name	Globus Version	Betriebssystem	CPU	CPU Cache	RAM
1	globus408test	4.0.8	(VM) Ubuntu 10.10	Intel Core2Duo, 2.93 GHz	6MB	1GB
2-5	astrodata01-astrodata04	4.0.8	Scientific Linux 5.5	2x AMD Opteron 250, 2.4GHz	1MB	8GB

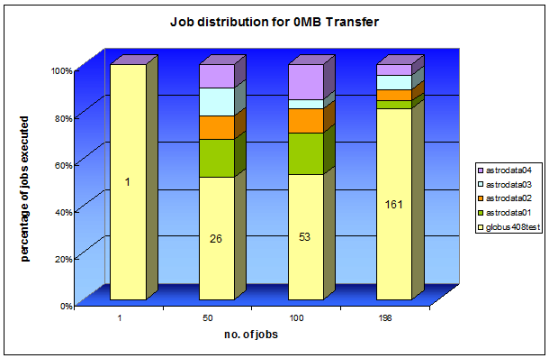
Tabelle 2: Verwendete CEs bei den Datentransfertests

Bei dieser Testreihe wurden nur Berechnungsjobs durchgeführt. Hier wurden also keine Eingabe- und

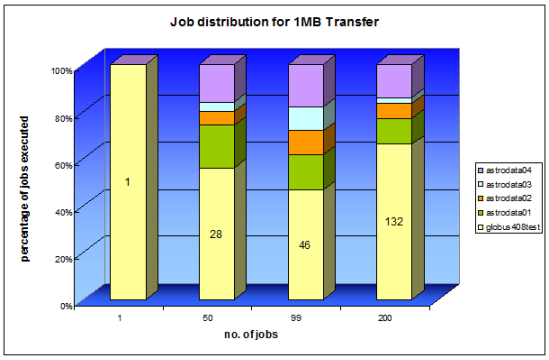
Ausführungsdateien benötigt, sondern nur jeweils lokal eine Berechnung der Zahl π durchgeführt und das Ergebnis zurück übertragen. Um den Effekt der Datenübertragung zu testen, wurden den Jobs Testdateien als „Ballast“ mitgegeben, welche GridWay also mit zum ausführenden Rechner übertrug, ohne diese Dateien dann jedoch bei der Jobbearbeitung zu benutzen. Im Detail wurden in den verschiedenen Jobsequenzen folgende Parameter variiert:

1. Die Anzahl der Jobs: 1, 50, 100, 200
2. Die Größe der bei jedem Job übertragenen Testdatei: 0MB, 1MB, 10MB

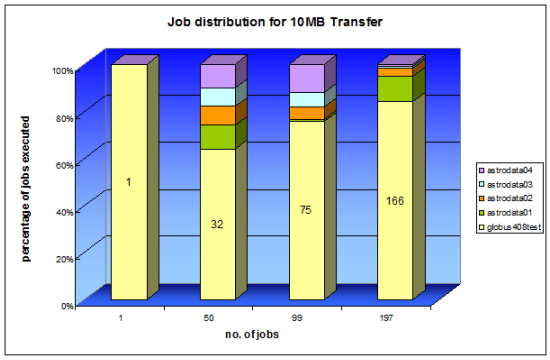
Als job-initiierender Rechner wurde in allen Fällen Rechner 1 verwendet. Die Ergebnisse sind in den Grafiken in den Abbildungen 6 bis 8 zusammengefasst. Abbildung 6 zeigt hierbei die Verteilung der Jobs über die 5 Compute Entities in Abhängigkeit von Transferdateigröße und Jobanzahl. In Abbildung 7 sieht man die von GridWay gemessene durchschnittliche Jobdauer (also die akkumulierte Transfer- und Ausführungszeit), während Abbildung 8 die tatsächliche durchschnittliche Jobdauer zeigt.



(a) Keine Transferdatei



(b) Transferdateigröße 1MB



(c) Transferdateigröße 10MB

Abbildung 6: Verteilung der Jobs über die Compute Entities unter Variation von Transfergröße und Jobanzahl

Bei der Verteilung der Jobs fällt zunächst auf, dass nur in einem Fall mehr als die Hälfte der Jobs auf andere Compute Entities als den job-initiierenden Rechner verteilt wurden, nämlich bei der Transferdateigröße 1MB und der Jobanzahl 100. Für diese Konstellation ergaben sich auch die höchsten Zeiten im Vergleich zu den Tests, bei denen Dateigröße oder Jobanzahl verändert wurden.

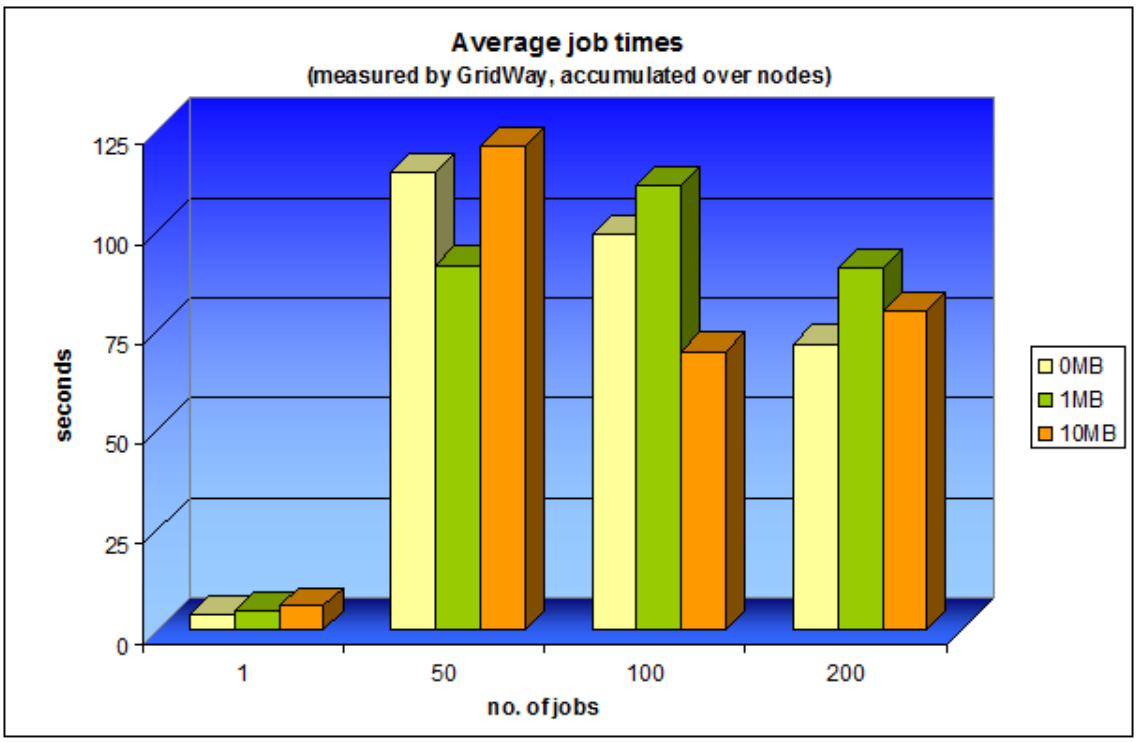


Abbildung 7: Durchschnittliche von GridWay gemessene, akkumulierte Jobzeit

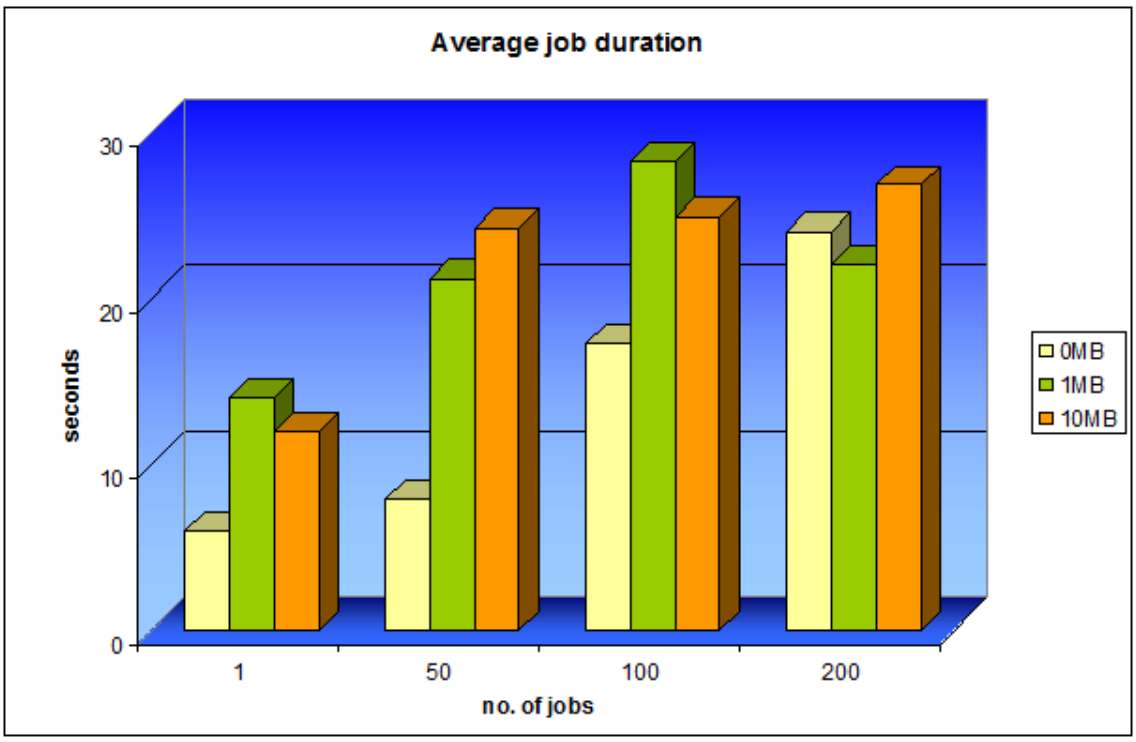


Abbildung 8: Durchschnittliche reale Jobdauer

Höhere von GridWay gemessene Jobzeiten ergaben sich nur die Fälle von 0MB und 10MB Dateigröße bei jeweils 50 Jobs, welches auch beides Fälle mit relativ hoher Jobverteilungsrate darstellen.

Hieraus läßt sich zunächst ableiten, dass eine höhere Verteilungsrate und eine längere durchschnittliche akkumulierte Jobdauer tendenziell zusammenfallen. Die tatsächliche Jobdauer verlängerte sich jedoch nicht im gleichen Maße mit erhöhter Jobverteilung, sondern sank teilweise sogar. Dies bestätigt die bereits im ersten Test gezogenen Schlussfolgerungen, dass die Verteilung von Jobs generell die Gesamtdauer deutlich verkürzt, auch wenn dies von GridWay selbst so nicht gemessen wird. Die Gründe für die erhöhte akkumulierte Jobdauer liegen, wie ebenfalls im ersten Test festgestellt, höchstwahrscheinlich am Kommunikations-Overhead von GridWay, welcher natürlich bei stärker verteilter Ausführung ansteigt.

Das erwartete Gleichgewicht zeichnete sich bei den Tests nur sehr rudimentär ab. Während die von GridWay gemessene Jobzeit mit steigender Anzahl an Jobs eher sank, stieg die tatsächliche Jobdauer leicht an. Dies liegt vermutlich darin begründet, dass bei einer größeren Anzahl Jobs die Verteilung über die Compute Entities sank und somit die von GridWay mitgerechneten Kommunikations- und die sich überlappenden Transferzeiten nicht mehr so stark ins Gewicht fielen.

Bemerkenswert ist dann auch genau diese mit wachsender Jobanzahl sinkende Verteilung der Jobs, und der damit einhergehende Anstieg der tatsächlichen Jobdauer insbesondere in den Fällen ohne Transferdatei („0MB“). Die Gründe hierfür liegen vermutlich in der bei höherer Jobanzahl erhöhten Wahrscheinlichkeit für Kommunikationsverzögerungen, durch welche die Jobs auf entfernten Rechnern vom Scheduler zu langsam als erledigt erkannt wurden. Hierdurch erfolgte somit eine verstärkte Zuweisung der Jobs an den job-initiierenden Rechner, die Parallelität der Job-Ausführung ging zu einem gewissen Teil verloren und führte so zu einem Anstieg der Gesamtdauer.

5.3 Auswertung

Die Ergebnisse der Scheduling-Tests lassen eine generelle Empfehlung für oder gegen eine verteilte Dienstausführung nur bedingt zu. Zusammenfassend lässt sich jedoch feststellen, dass bei einer durchschnittlichen Netzwerkgeschwindigkeit der Transfer von Dateien in der Größenordnung bis zu 10 MB nur einen geringen Performancenachteil bedeutet und die Vorteile der Parallelisierung durch die verteilte Ausführung hier deutlich überwiegen. Insbesondere bei einer hohen Anzahl (¿100) von Jobs, wie sie zum Beispiel im in Abschnitt 7 beschriebenen Anwendungsfall anfallen, ist somit klar eine verteilte Ausführung anzuraten. Im Idealfall (Dateitransferdauer pro Job 1-2 Sekunden) kann hierbei eine annähernd linear-reziproke Verringerung der Gesamtdauer einer Jobsequenz erreicht werden.

Um genaue Aussagen bezüglich des Verhältnisses von Netzwerkgeschwindigkeit, Dateigröße, Anzahl der Compute Entities und Jobdauer zu treffen, wären noch weitergehende Tests notwendig, welche im Rahmen von WissGrid jedoch nicht mehr durchgeführt werden konnten. So wäre z.B. noch genauer zu untersuchen, wie sich die Anzahl an und Verbindungsgeschwindigkeit zu den Compute Entities auf die Transfergeschwindigkeit und die Ausführungsdauer auswirkt. Auch die Größe der transferierten bzw. zu bearbeitenden Dateien sollte in weiteren Tests noch in den dreistelligen MB-Bereich, evtl. auch in den GB-Bereich erhöht werden. Des weiteren könnte eine höhere Anzahl an Jobs abgeschickt werden, wenn die GridWay-Einstellungen dahingehend angepasst würden, und somit die Verteilung der Jobs in Abhängigkeit von Dateigröße und Jobdauer noch besser untersucht werden.

Das größte Hindernis für interessierte Nutzer und Communities bei der Umstellung auf eine verteilte Dienstausführung dürfte jedoch wohl darin bestehen, die job-initiierende Umgebung einzurichten und zu konfigurieren sowie die Erreichbarkeit, Verfügbarkeit (Installation und Konfiguration) der Dienste auf den Compute Entities und die möglichst reibungslose Kommunikation mit diesen sicherzustellen. Um die Performance und die Funktionalität der verteilten Dienstausführung ohne großen Aufwand testen zu können, wurde von WissGrid eine virtuelle Maschine eingerichtet, in welcher alle notwendige und im Rahmen dieser Tests verwendete Software bereits installiert und vorkonfiguriert wurde (s. auch Abschnitt 6.4).

6 Prototypische Implementierung

Die vom WDF zu erfüllenden Funktionalitäten des Schedulings und des Repository-Zugriffs wurden im Rahmen von WissGrid mit dem Scheduler GridWay und der Globus-Middleware prototypisch umgesetzt. In diesem Abschnitt wird beispielhaft die GridWay Job-Submission eines einfachen Dienstaufrufs (Formatvalidierung mit JHOVE2) beschrieben, wobei die Daten aus dem WissGrid-Repository [13] bezogen werden und die Ergebnisse der Formatvalidierung in das Repository zurückgeschrieben werden. Die Job-Ausführung erfolgt auf Compute Entities mit der Globus Middleware, Version 4.0.8.

Im Folgenden werden bzgl. der Job-Submission zwei Ansätze verfolgt: per Job-Template und per DRMAA (Distributed Resource Management Application API). Bei ersterem Ansatz wird ein vordefiniertes GridWay Job-Template für die Job-Submission eingesetzt, bei letzterem wird das Job-Template programmgesteuert generiert.

Um den API-gestützten Ansatz nachvollziehen zu können, wird im Folgenden zunächst die Einrichtung der Arbeitsumgebung erläutert. Anschließend werden die beiden Ansätze und die dafür eingesetzten Skripte und Programme vorgestellt. So soll eine Basis zur Erfüllung allgemeiner benutzerspezifischer Anforderungen geschaffen werden, welche für den jeweils auszuführenden Dienst angepasst bzw. realisiert werden kann.

6.1 Anforderungen und Benutzerumgebung

GridWay DRMAA Java Bibliothek erzeugen

Um die Job-Submission über GridWay per DRMAA API absetzen zu können, muss die zugehörige Java-Bibliothek im Rahmen der GridWay-Kompilierung (Build) mit erzeugt werden, was wie folgt erreicht wird:

```
./configure --prefix=/opt/gridway --enable-debug --with-tests \
      --enable-drmaa1-java
make clean
make
make install
```

Die Option `enable-drmaa1-java` bewirkt, dass die Java DRMAA 1.0 Bibliothek generiert und im Zielverzeichnis /opt/gridway/lib/ abgelegt wird. Bei Erzeugung oder Ausführung von Java-Klassen muss der Java Laufzeitumgebung der Library-Path (-Djava.library.path=/opt/gridway/lib/) bekannt gegeben werden. Zudem muss das zugehörige Java-Archiv /opt/gridway/lib/drmaa.jar und die Bibliothek zum Apache Command Line Interface (CLI) in den Classpath aufgenommen werden, was hier jeweils durch maven sichergestellt wird.

Konfiguration von eclipse und maven

Um mit eclipse und maven zu arbeiten, sind folgende Einstellungen vorzunehmen:

- Erzeugung und Ausführung mit eclipse:

Run Configuration -> Arguments ->
Program arguments:
`-Djava.library.path=/opt/gridway/lib/`
VM arguments:
`- -file \s/README.txt, s/LICENSE.txt ,s/Copyright.txt\`

Run Configuration -> JRE -> `-Djava.library.path=/opt/gridway/lib/`

CLASSPATH -> User Entries -> Maven Dependencies

- maven pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.gridway</groupId>
    <artifactId>drmaa</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>commons-cli</groupId>
    <artifactId>commons-cli</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
```

Installation von JHOVE2 und Wissgrid-Repository

Damit die Ausführung eines JHOVE2-Jobs durchgeführt werden kann, ist es erforderlich, dass JHOVE2 auf den verwendeten CEs installiert ist und dass das JHOVE2 bin-Verzeichnis in der PATH Umgebungsvariable aufgeführt ist, damit JHOVE2 ohne Kenntnis des konkreten Installationspfades ausgeführt werden kann. Selbiges gilt auch für die WissGrid-Repository Komponenten. iRODS, Fedora, Solr und die wissgridservice-Komponente müssen installiert und gestartet sein, damit das Laden von iRODS Daten-Objekten und das Zurückschreiben von JHOVE2-Ergebnissen ins Repository durchgeführt werden kann. Einzelheiten hierzu können der WissGrid Dokumentation „Realisierung der WissGrid-Spezifikation: Grid-Repository“[13] entnommen werden.

6.2 Job-Submission per Job-Template

Bei der Job-Template-basierten Job-Submission enthält das Job-Template u.a. Angaben darüber *was*, *womit*, *auf welchen potenziellen CEs* ausgeführt werden soll. In dem in Listing 1 angegebenen Job-Template, legt die Option `EXECUTABLE` fest, dass das angegebene Shell-Skript (jhove2_irods.sh, siehe Listing 2) ausgeführt werden soll. Dieses Shell-Skript ist in `INPUT_FILES` aufgeführt, was bedeutet, dass es von GridWay bei der Submission mit nach Globus übertragen und in einem job-spezifischen, temporären Verzeichnis abgelegt wird. Die Option `ARGUMENTS` enthält Aufrufparameter für das Shell-Skript, welche dem Shell-Skript bei seiner Ausführung übergeben werden:

1. Name der JHOVE2 Ergebnisdatei.

2. Absoluter iRODS-Pfad des zu bearbeitenden Daten-Objekts in iRODS (kann auch relativ zum *current working directory* in iRODS angegeben werden).

3. Name der Datei, die von JHOVE2 verarbeitet werden soll, d.h. das aus iRODS bezogene und in Globus (temporär) gespeicherte Daten-Objekt.

4. Name unter dem das JHOVE2-Ergebnis in das Repository geschrieben werden soll.

Die Option `REQUIREMENTS` enthält hier lediglich Namen von CEs, die für die Job-Ausführung in Betracht gezogen werden sollen. Bezüglich weiterer Optionen und deren Verwendung sei auf die GridWay Benutzerdokumentation [11] verwiesen.

Listing 1: Job-Template: jhove2_irods.jt

```
EXECUTABLE=./jhove2_irods.sh
ARGUMENTS=out /tempZone/home/rods/multi/a/data/postgresql-9.0.3.tar
    postgresql-9.0.3.tar jhove2.out.xml
INPUT_FILES=jhove2_irods.sh

REQUIREMENTS=HOSTNAME="globus408test.sub.uni-goettingen.de" |
    HOSTNAME="tglobe.sub.uni-goettingen.de"
RESCHEDULING_INTERVAL=0
RESCHEDULE_ON_FAILURE=no
NUMBER_OF_RETRIES=0
```

Dieses Job-Template bildet die Basis für die Job-Submission über GridWay. Intern wird das Template in ein für Globus interpretierbares Job Submission Description Language (JSDL) Dokument konvertiert, und es wird eine Globus-spezifische Job-Submission durchgeführt. Wenn die Submission erfolgreich durchgeführt wurde und alle Input-Files auf Globus vorliegen, wird das Executable mit den Parametern (Option *ARGUMENTS*) aufgerufen und ausgeführt. Das Shell-Skript und eine Erläuterung der wesentlichen Zeilen findet sich in Anhang D.1.

Eine Schwachstelle des Job-Template-basierten Ansatzes ist, dass mit jedem vorliegenden Job-Template nur das Daten-Objekt, welches in der Option ARGUMENTS angegeben ist, bearbeitet werden kann. Es ist zwar denkbar, auf diesem Wege eine Liste mit iRODS Daten-Objekten oder ein iRODS-Verzeichnis zu bearbeiten, dennoch sind Anpassungen am Job-Template erforderlich. Der im nachfolgenden Abschnitt beschriebene Ansatz schafft hier mehr Flexibilität, indem die zu bearbeitenden Dateinamen als Aufruf-Parameter an ein Java-Programm übergeben werden.

6.3 Job-Submission per API

Bei der API-basierten Job-Submission wird das Job-Template nicht statisch festgelegt, sondern programmatisch erzeugt, und auch die Job-Submission erfolgt per API-Funktion. Das ausführbare Shell-Skript aus dem Job-Template-basierten Ansatz wird unverändert weiterverwendet.

Beim Programm-Aufruf muss die Option *-f* bzw. *–file*, gefolgt von einer komma-separierten Datei-Liste mit relativen oder absoluten iRODS-Pfaden (`<file>[,file]*`) angegeben werden. Ausgangspunkt der Programmausführung ist die Main-Klasse *GW_JobSubmission*, welche zunächst die

Kommandozeilen-Argumente abarbeitet und dann die eigentliche Verarbeitung veranlasst. Hierbei wird zu jeder Datei ein Job-Template erzeugt, die zugehörige Job-Submission veranlasst und jeweils einen Thread gestartet, welcher den Status der Ausführung nach Beendigung des Jobs abruft und eine Statusmeldung ausgibt (Klasse JobAgent). Die generierten Job-Templates entsprechen dem in Listing 1 in Anhang D.1, nur wird hier der Dateiname der generierten Job-Templates vom Dateinamen des zu verarbeiteten Daten-Objekts abgeleitet. In Anhang D.2 sind die Quellcodes der beiden Klassen aufgelistet und die wesentlichen Stellen besprochen.

6.4 Virtuelle Testumgebung

Im Rahmen von WissGrid wurde eine virtuelle Maschine erstellt, in welcher bereits alle verwendeten Komponenten vorinstalliert und -konfiguriert sind:

1. Globus Version 4.0.8
2. GridWay Version 5.8
3. JHOVE2
4. FITS
5. iRODS
6. Fedora
7. Solr
8. wissgridservices

Dieses virtuelle Festplattenimage kann zu Testzwecken genutzt werden, um die verteilte Job-Ausführung und die Anbindung des Schedulers an das WissGrid-Repository zu testen, ohne die gesamten Installations- und Konfigurationsschritte durchlaufen zu müssen. Das Image, eine Dokumentation der Installation inklusive Hinweise zur Benutzung finden sich auf der WissGrid-Webseite `www.wissgrid.org`.

7 Anwendungsfall

Als Anwendungsfall wurde im Rahmen von WissGrid der sogenannte TextGrid Masseningest betrachtet. Auf Seiten von WissGrid stellte sich jedoch die Auswahl und Konfiguration eines geeigneten Schedulers als problematisch dar, während bei TextGrid Unklarheiten bzgl. der zu nutzenden Technologie sowie einer repräsentativen Grid-Struktur zu Verzögerungen führten. Daher wurde dieser Anwendungsfall nicht abschließend umgesetzt. Mit der prototypischen Implementierung des WDF (siehe Abschnitt 6) lassen sich jedoch alle Teilschritte der nachfolgenden, im Projektverlauf gemeinsam ausgearbeiteten Spezifikation durchführen.

7.1 TextGrid Masseningest

Bei diesem Anwendungsfall sollen über einen gewissen Zeitraum erstellte Digitalisate des Göttinger Digitalisierungszentrums (GDZ) im Paket in das Grid-Repository geschrieben werden. Hierbei ist eine Formatvalidierung und -extrahierung durch den Dienst JHOVE2 durchzuführen, und die Ergebnisse dieses Dienstes sind mit den Repository-Daten zu verknüpfen. Als minimal ausreichende Performance wird hierfür definiert, ein Paket von der Größe der in einem Monat anfallenden Daten innerhalb eines Monats zu verarbeiten und im Repository zu speichern. Detailliertere Spezifikationen zu den in diesem Anwendungsfall verwendeten Daten finden sich in Anhang A.

Workflow

Im Folgenden werden die erforderlichen Schritte des Anwendungsfalles beschrieben. Dabei ist hier nur der Ablauf nach Ingest Variante A (s. Abschnitt 7.1) aufgelistet, Abweichungen für Variante B sind durch Fußnoten kenntlich gemacht.

1. **Submit:** Auslösen des Ingest Vorganges durch das GDZ
   a) Annahme des Ingest-Vorgangs durch den TextGrid Server
   b) Transfer der Daten vom GDZ zum TextGrid-Server*
   c) Modellierung der angelieferten Daten gemäß dem TextGrid-Objektmodell bzw. den unterstützten Import-Profilen (z.B. DFG-Viewer METS)*
2. **Lokale Verarbeitung I:** *
   a) Überprüfung der Metadaten
   b) Auflösen der Referenzen zwischen Dateien
3. **Ingest:** Schreiben der Daten in den Grid-Speicher*
   a) Schreiben in iRODS Server
   b) ggf. Benachrichtigung und Aktualisierung von Fedora mittels Callback
4. **Dienstausführung** Formatvalidierung sowie Extrahierung der tech. Metadaten mit JHOVE2
   a) **Identify:** Ermittlung von verfügbaren CEs, auf welchen der gefragte Dienst installiert ist, einschließlich

i. (geschätztem) Rechenaufwand
ii. Verfügbarkeit von (temporärem) Speicher
iii. (erwartetem) Transferaufwand

und darauf basierend Auswahl der CEs und ggf. Aushandeln/Reservieren von Rechenzeit

b) **Extract:** Extrahieren der benötigten Daten aus dem Repository und Aufteilen in geeignete Pakete†
c) **Stage-In:** Transfer der Daten auf die CEs†
d) Ausführung des Dienstes (JHOVE2) auf den jeweiligen CEs
e) **Stage-Out** Annahme des Ergebnisses und Weiterleitung zur Verarbeitung (Schritt 5b)

5. **Lokale Verarbeitung II:**
   a) Vergabe von PIDs (Persistent Identifier)
   b) Aktualisieren der Metadaten
   c) Erstellung des Suchindexes
   d) Aktualisieren der Rechte
6. **Notify:** Benachrichtigung über Abschluß des Ingest-Vorgangs bzw. über evtl. aufgetretene Probleme

Insgesamt ist das WDF für die Aufgaben in Schritt 4 zuständig. Weiterhin wird davon ausgegangen, dass der verwendete Scheduler, wie in Abschnitt 4.1 beschrieben, mindestens die Schritte 4c, 4d und 4e initiiert und koordiniert. Alle weiteren Schritte werden entweder vom TextGridServer, Fedora oder iRODS übernommen.

Mögliche Ingest-Verfahrensweisen im Anwendungsfall TextGrid Masseningest

Je nach Struktur des Repositories und Lage der Dienste und Daten lassen sich zwei generelle Varianten des Ingests unterscheiden, welche im Diagramm in Abbildung 9 dargestellt sind und im Folgenden erläutert werden. Rote Linien kennzeichnen hierbei Kommunikationswege zwischen Komponenten, bei denen der Transfer von in das Repository einzuspielenden Daten stattfindet.

Ingest Variante A: Der Fokus dieser Ingest-Variante liegt auf dem initiierenden Submit(1) an TextGrid CRUD (Create, Retrieve, Update, Delete), welches direkt den Fedora-Server anspricht. Fedora prozessiert zunächst die erhaltenen Daten(2) durch Überprüfung der bereits enthaltenen Metadaten sowie durch Auflösen von eventuellen Referenzen zwischen den Dateien. Wenn möglich parallel hierzu startet Fedora die Anfrage(4) an das WDF zur Identifizierung(4a) der für die angefragten/anfallenden Dienste in Frage kommenden CEs. Nach erfolgter lokaler Verarbeitung beginnt Fedora mit dem sequentiellen Ingest in iRODS (3a). Nach erfolgreichem Ingest (oder evtl. auch schon parallel hierzu) erfolgt dann durch den Scheduler des WDF das Extrahieren(4b)

*Bei der Ingest Variante B (s. Abschnitt 7.1) findet der Ingest nach iRODS bereits vor der **Lokalen Verarbeitung I** statt. Sämtliche weiteren Schritte werden dann per Callback an Fedora oder über ein Messaging-System — ausgehend von iRODS — angestoßen.

†Die Schritte 4b und 4c sind je nach Verteilung der Daten auf den CEs nur bedingt erforderlich, was in Abbildung 9 durch die gestrichelten Linien verdeutlicht wird.

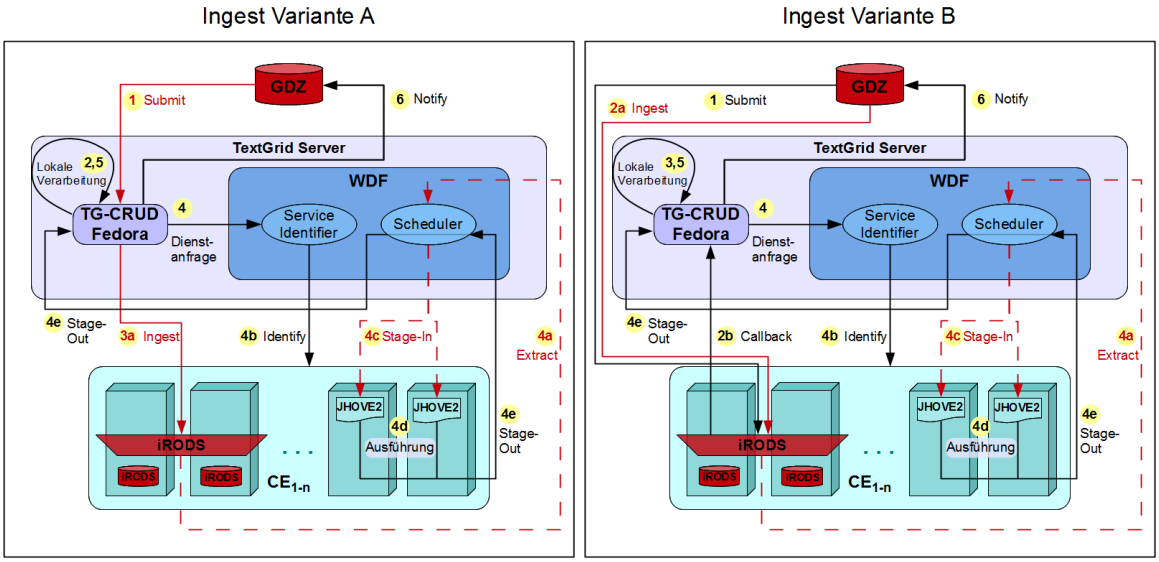


Abbildung 9: Mögliche Ingest Varianten im Anwendungsfall

der Daten aus dem iRODS-Repository, das Stage-In(4c) zu den identifizierten CEs, das dortige Verarbeiten(4d) der Daten durch die Dienste sowie das Stage-Out(4e) der Ergebnisse an Fedora und die abschließende Verarbeitung der Dienstergebnisse(5) sowie die Aktualisierung der Rechte. Abschließend wird das GDZ über den Abschluß benachrichtigt(7).
Zusammenfassend läßt sich diese Variante wie folgt beschreiben:

1. GDZ initiiert Ingest über TextGrid-CRUD an Fedora
2. Fedora leitet Ingest-Anfrage an WDF weiter, wo Formatvalidierung wie unter 7.1 beschrieben durchgeführt wird
3. Parallel hierzu werden Daten sequentiell im iRODS-Server gespeichert

Ingest Variante B: Bei dieser Ingest-Variante erfolgt der Submit(1) und sowie Ingest(2a) unter Umgehung von Fedora direkt an den iRODS-Server. Mittels Callback-Aufrufen(2b) wird nach dem Ingest (oder parallel hierzu) Fedora über die neuen Daten informiert und initiiert parallel zur lokalen Verarbeitung(3) wiederum die Identifizierung(4a-e) und Abarbeitung der Dienste, wie bereits unter **Ingest A** beschrieben. Die abschließenden Schritte (5-7) entsprechen ebenfalls der Ingest Variante A.

1. GDZ initiiert Ingest direkt in iRODS-Server
2. iRODS informiert Fedora per Callback über Ingest
3. Fedora beginnt mit Verarbeitung der vorhandenen Metadaten und stößt parallel hierzu Identifizierung und Ausführung der Dienste an

Bei beiden Varianten ist prinzipiell die Ausnutzung von Lokalität möglich, dass also die Dienste auf genau den CEs ausgeführt werden können, wo die Daten bereits vorhanden sind, um den erforderlichen Datentransfer zu reduzieren. Die Schritte **Extract** und **Stage-In** sind daher nur

für die Fälle durchzuführen, in denen die Daten nicht auf den CEs liegen, wo die Dienste ausgeführt werden sollen. Hierfür werden allerdings in beiden Fällen Informationen über die physischen Speicherorte benötigt, welche entweder über Fedora oder direkt bei iRODS abgefragt werden müssen.

Für die Ingest Variante A spricht, dass sich durch die Vorverarbeitung von Fedora evtl. die auszuführenden Dienste und somit geeignete CEs für den Ingest (und somit die Ermöglichung lokaler Diensteverarbeitung) sinnvoller identifizieren lassen. Die Ingest Variante B hingegen hat ihre Vorteile darin, dass zum einen die von iRODS selbst angelegte Dateistruktur deutlich sinnvoller strukturiert ist als bei einem Ingest über Fedora, zum anderen der Ingest durch die direkte Speicherung in iRODS parallelisiert werden könnte, während Fedora jedes Objekt sequentiell abarbeitet. Welche Variante schließlich effizienter arbeitet, hängt zum Großteil von den Möglichkeiten zur Parallelisierung der anderen Arbeitsschritte ab, welche sich erst bei der Implementierung zeigen werden. Auch hier gilt somit, dass die Entscheidung über den Einsatz der Varianten letztendlich von jeder Community selbst getroffen werden muss und das WDF beide Varianten unterstützen sollte.

A Spezifikationen zum Anwendungsfall TextGrid-Masseningest

Kollektionen: Exemplarische Auswahl an Masterfiles des Göttinger Digitalisierungszentrum (GDZ)

Datenquelle: GOOBI (Software) + GWDG (Storage)

Datenvolumen: zunächst exemplarisch im einstelligen GB-Bereich; perspektivisch im Bereich 500+GB/Monat sowie ca. 100TB an bereits vorhandenen Bestandsdaten

Datenbeschaffenheit: TIFF-Dateien + METS/XML-Kurzbeschreibung + teilweise TEI-Volltextbeschreibung

Rechte in Bezug auf Ingest-Verarbeitung: bisher unproblematisch, da open access

Dienste: JHOVE2

Repository/Datensenke: Fedora-iRODS Server + GWDG Storage

Rechenressourcen: eigene Globusinstallation + evtl. verteilte D-Grid-Ressourcen

Grid-Knoten :

- Göttingen, GWDG
- evtl. später Würzburg, Uni (Replikation)

Zeitbegrenzung: bei Dateien mit ca. 2MB Größe, Gesamtvolumen 500GB:
200000 Dateien im Monat, also 1 Datei in 12 Sekunden

Parallelisierbarkeit: alle Dateien einzeln prozessierbar

Effizienz des bisherigen Verfahrens: noch zu testen

B Scheduler Übersicht

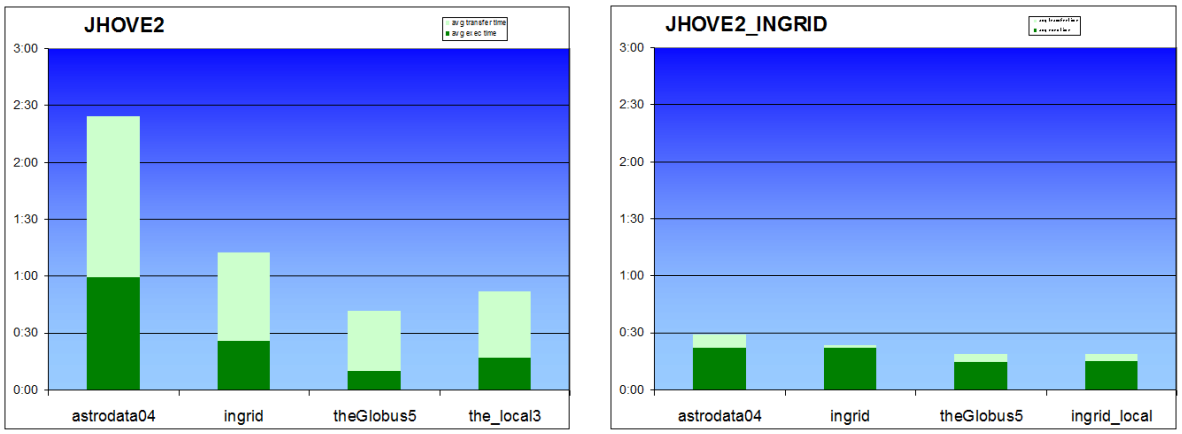
Die folgende Tabelle liefert eine komprimierte Übersicht über die Spezifikationen der in Abschnitt 4.2 beschriebenen Scheduler.

Bez.	GridWay	GWES	WSS	(WisNetGrid)	KOALA	CSF
Name	GridWay	Generic Workflow Execution Service	Workflow Scheduling Service	—	Processor and data co-allocation system	Community Scheduler Framework
Entwickler/ Projekt	Globus	Fraunhofer FIRST	C3Grid	WisNetGrid	TU Delft	Globus
URL	2	3	4	5	6	7
Grid-Kontext/ Middleware	Globus	Globus	Globus	—	JavaGAT	Globus
Beschreibungs-sprache(n)	JSDL	GWorkflowDL	WSDL	suprimePDL	Globus RSL	Globus RSL
Data awareness	Prototyp	nein	ja	nein	ja	(Interfaces)

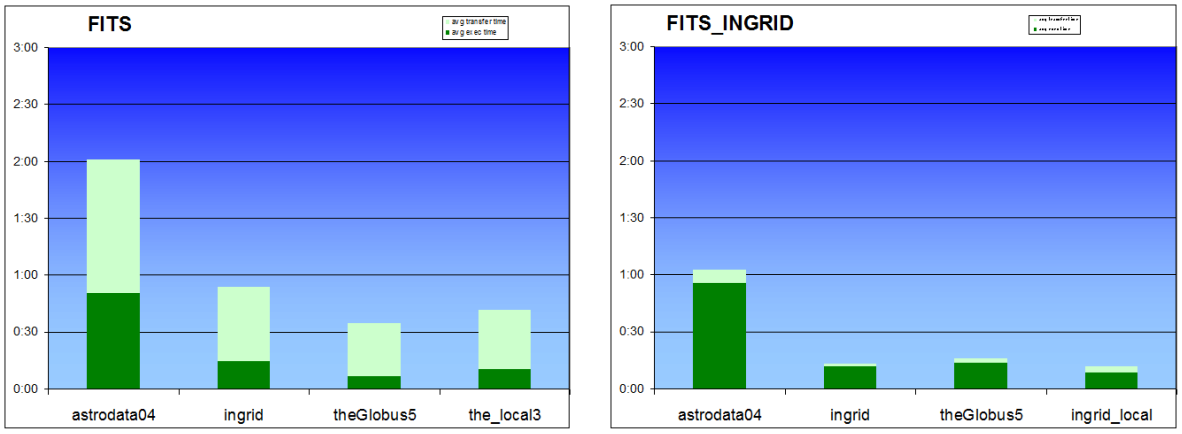
[b] http://www.gridway.org/
[c] http://www.gridworkflow.org/
[d] http://www.c3grid.de/
[e] http://www.wisnetgrid.org/
[f] http://www.st.ewi.tudelft.nl/koala/
[g] http://www.globus.org/toolkit/docs/4.0/contributions/csf/

C Ergebnisse der Scheduling-Tests

C.1 Scheduling-Testreihe zur Job-Verteilung

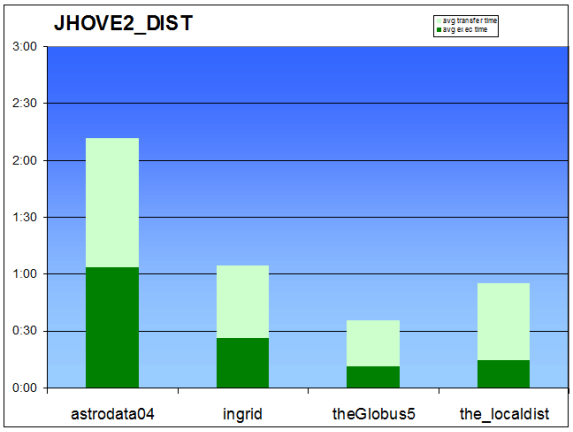


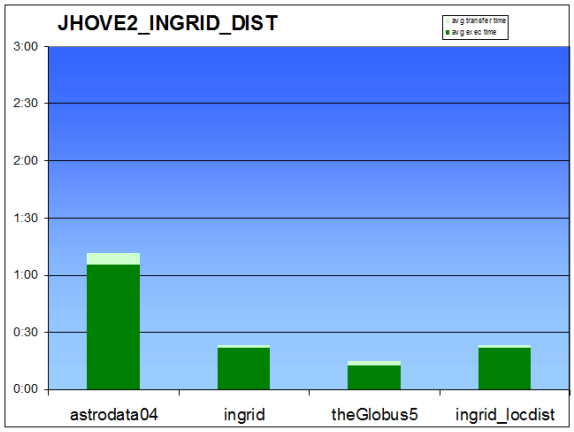
(a) JHOVE2-Jobs



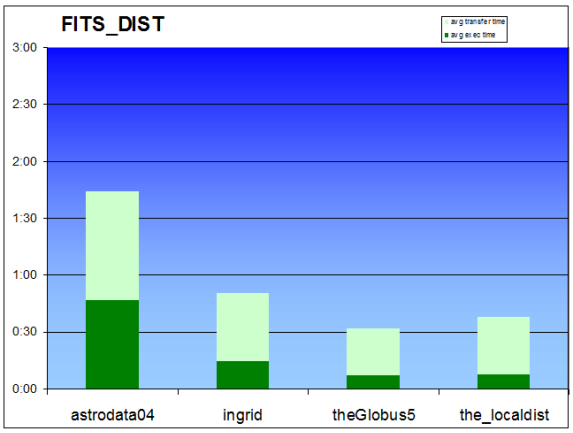
(b) FITS-Jobs

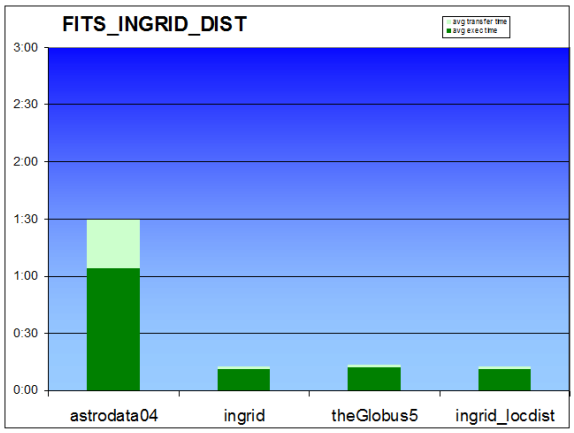
Abbildung 10: Durchschnittliche Job-Ausführungszeiten in Minuten für verschiedene Jobs mit Transfer von Eingabe- und Ausführungsdatei



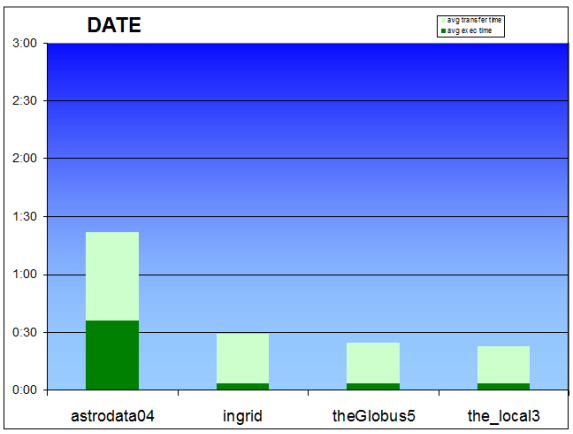


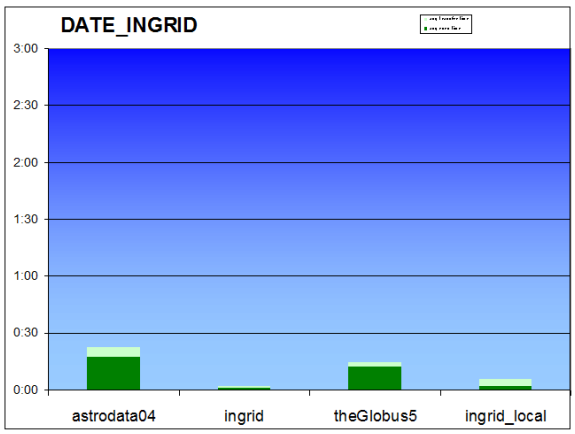
(a) JHOVE2-Jobs





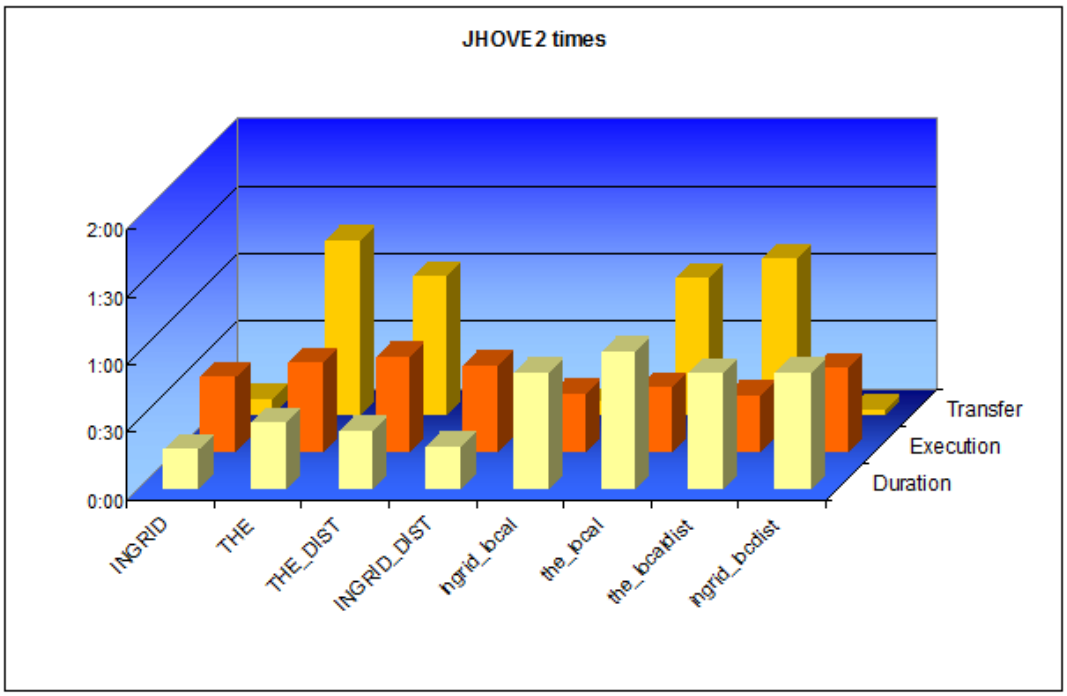
(b) FITS-Jobs



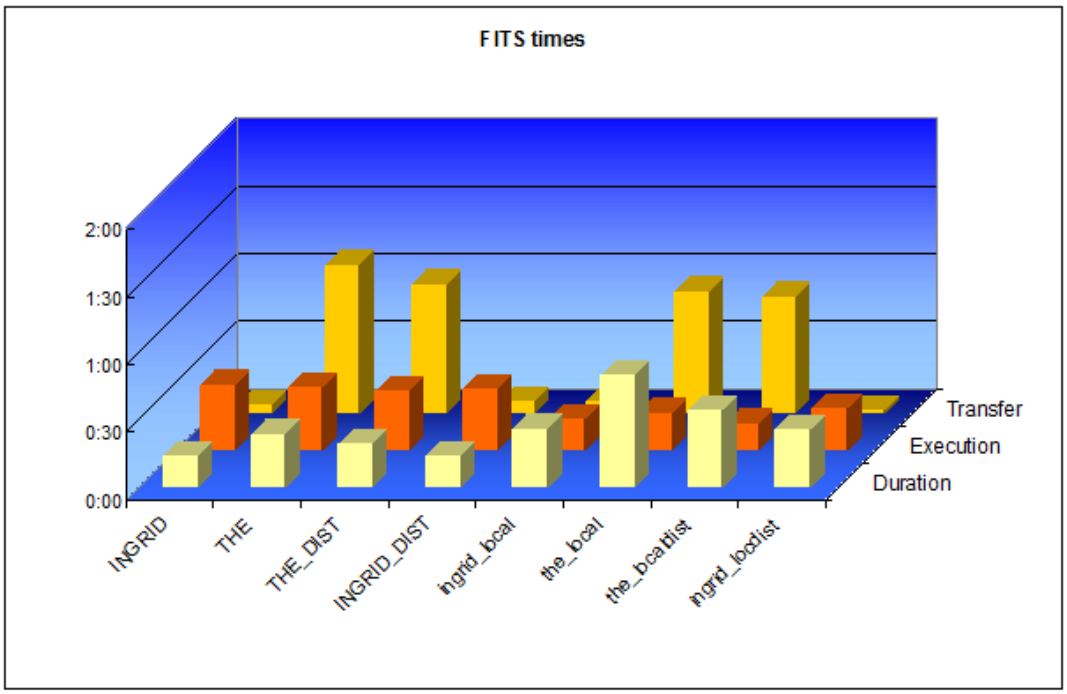


(c) DATE-Jobs

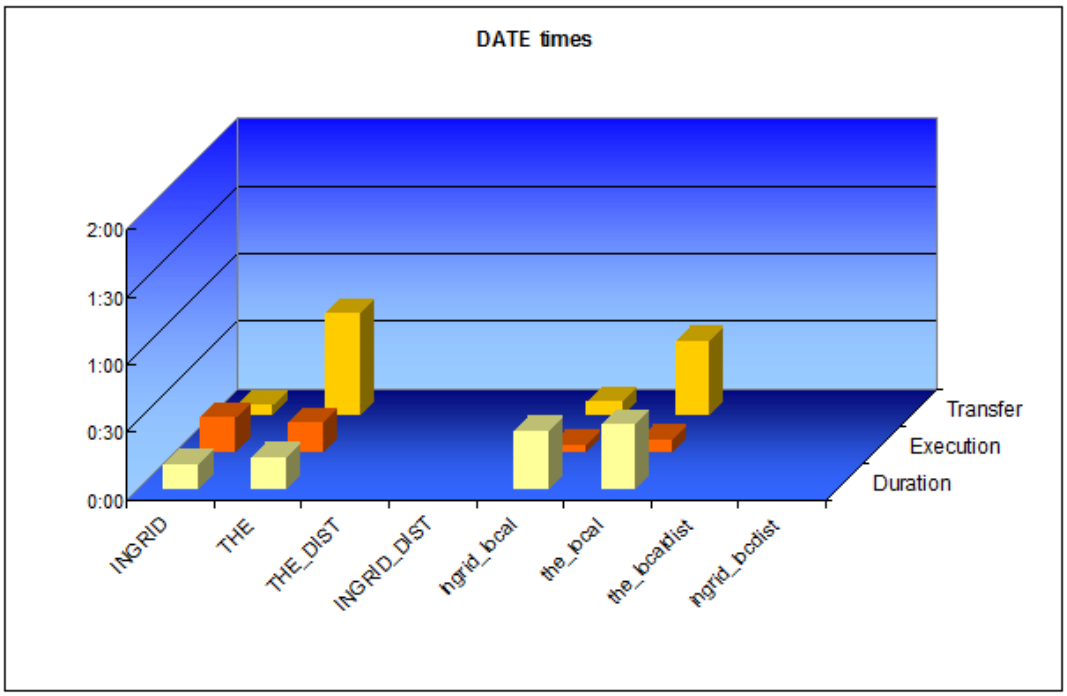
Abbildung 11: Durchschnittliche Job-Ausführungszeiten in Minuten für verschiedene Jobs ohne Transfer von Eingabe- und Ausführungsdatei



(a) JHOVE2-Jobs

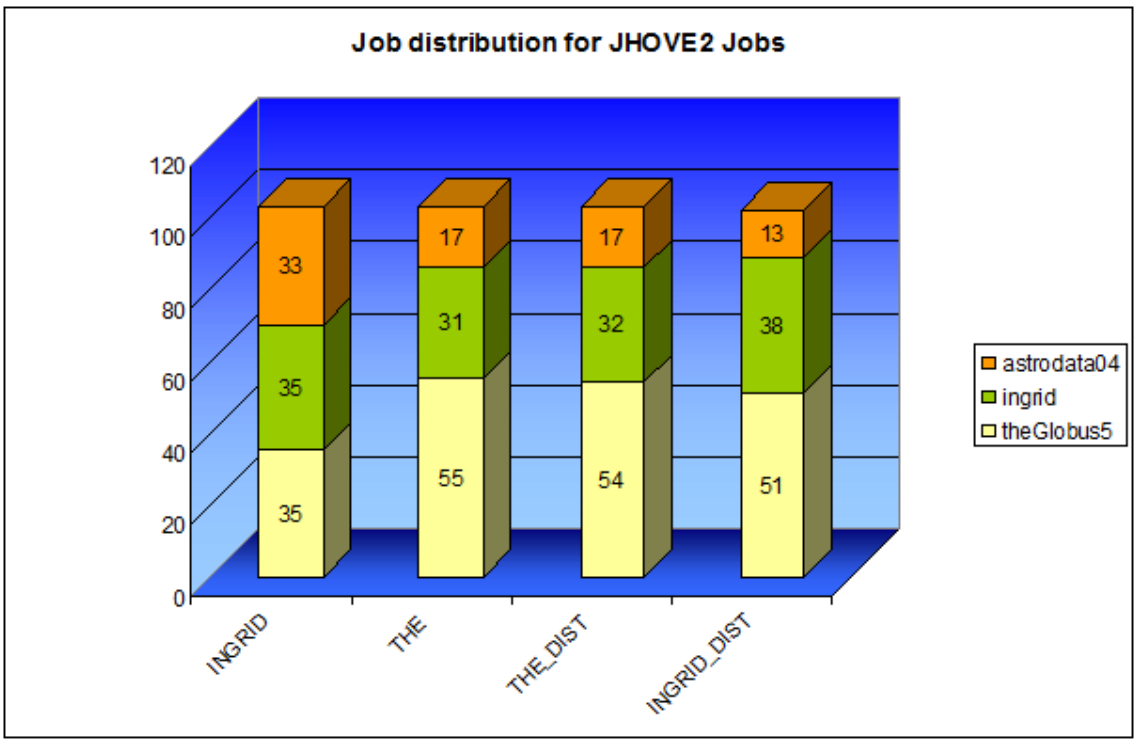


(b) FITS-Jobs

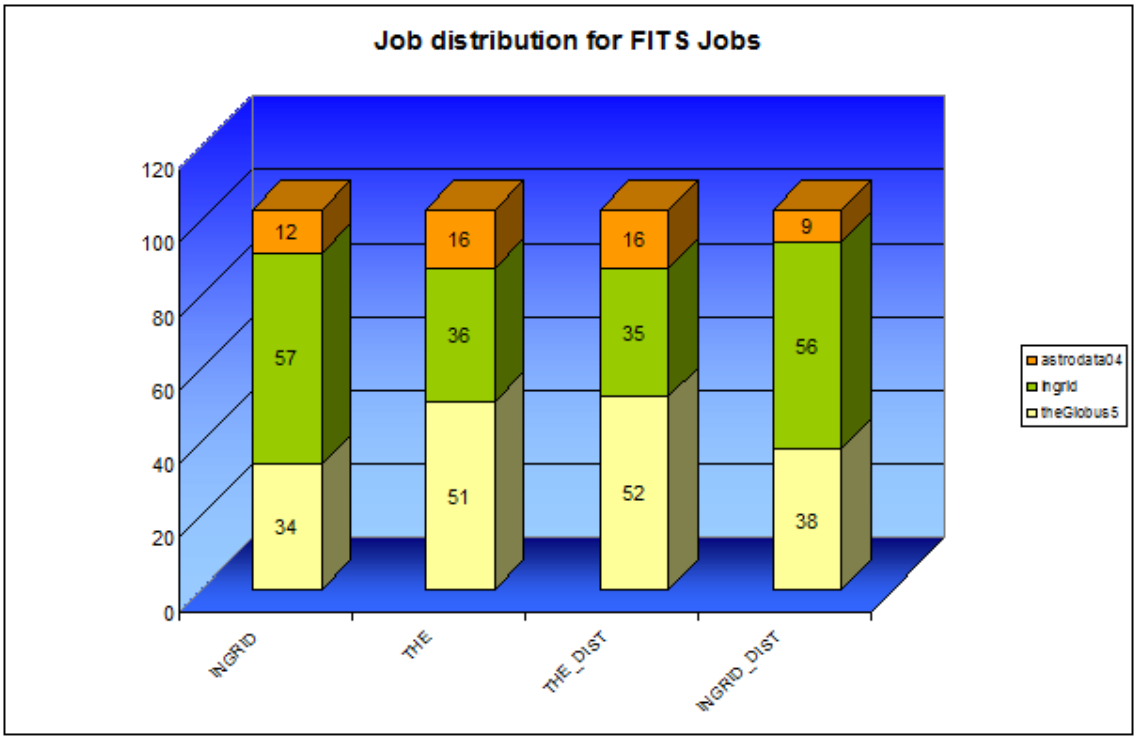


(c) DATE-Jobs

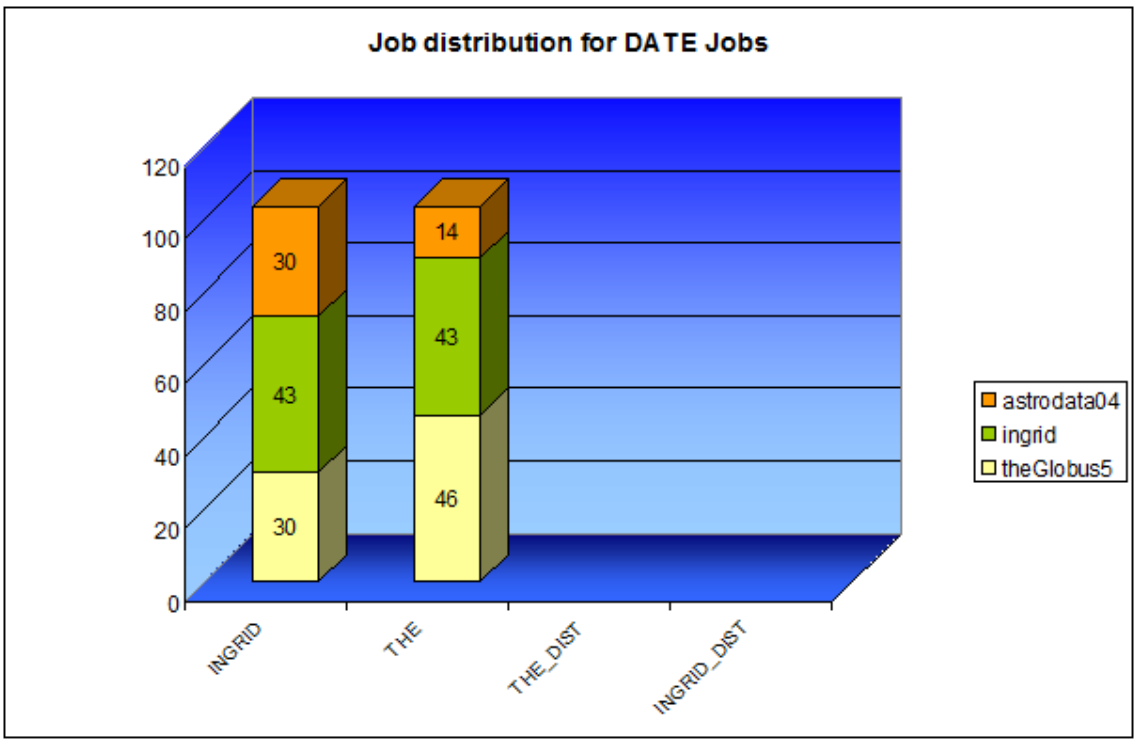
Abbildung 12: Durch GridWay gemessene Zeiten (in Stunden) von Jobsequenzen à 103 Jobs unter verschiedenen Konfigurationen, mit verschiedenen Jobs



(a) JHOVE2-Jobs



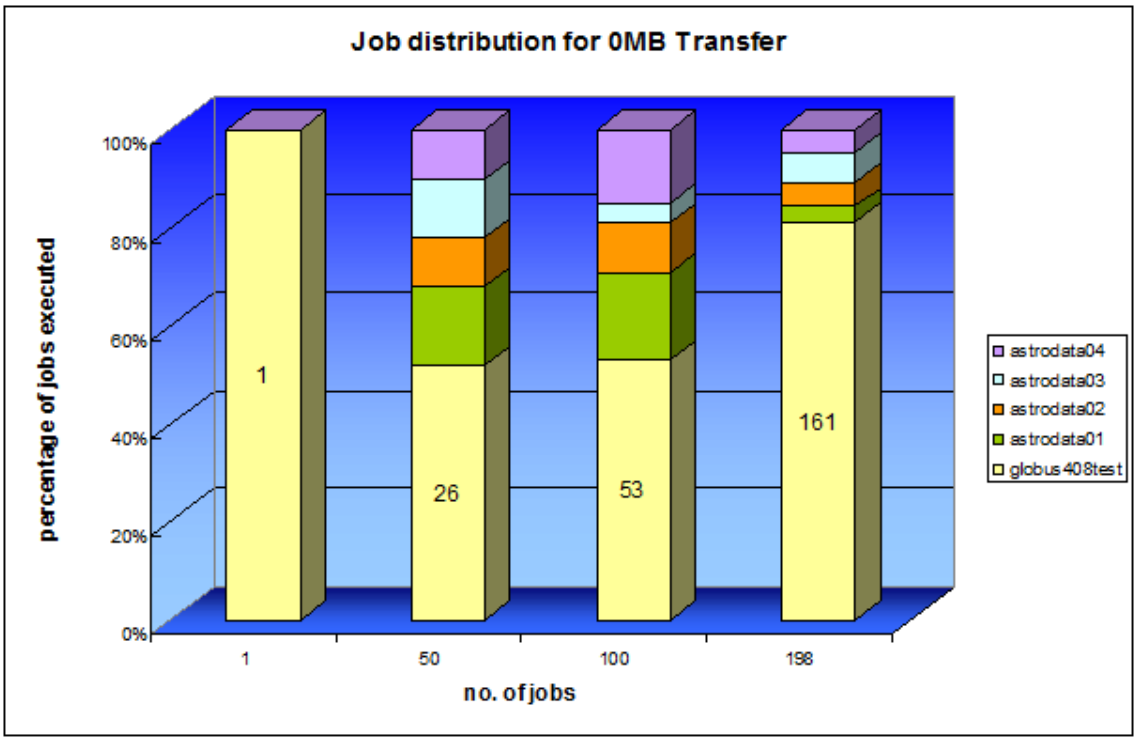
(b) FITS-Jobs



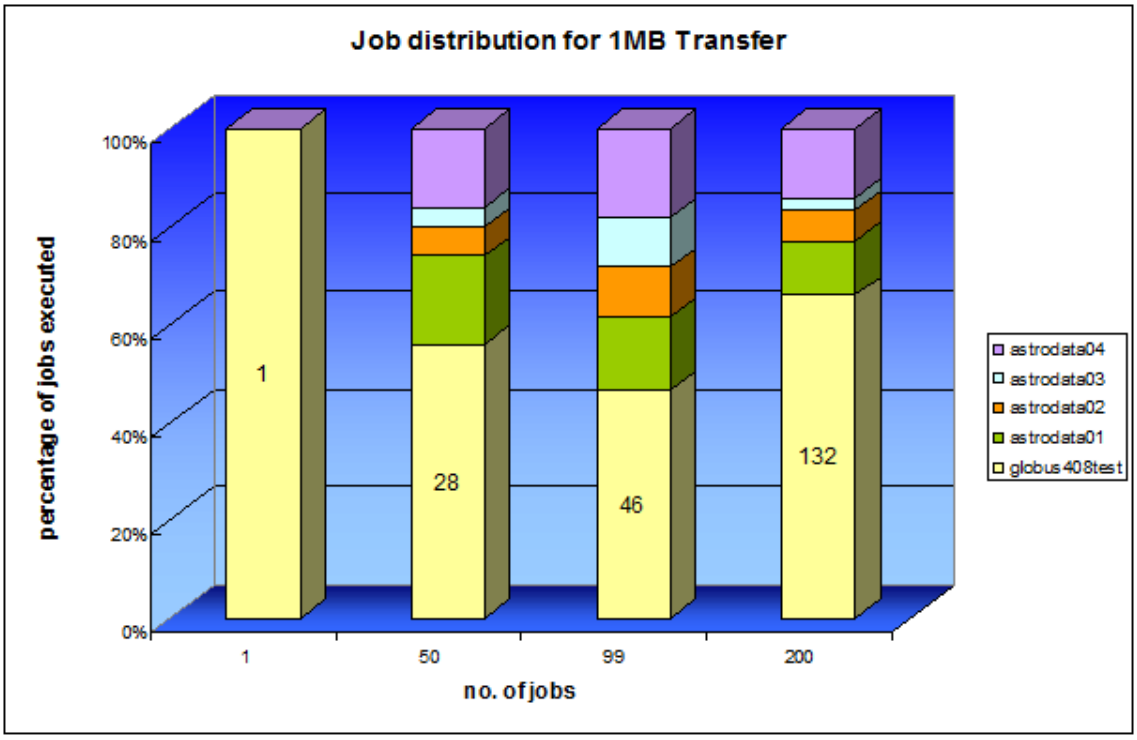
(c) DATE-Jobs

Abbildung 13: Job-Verteilung von 103 Jobs über verwendete Compute Nodes unter verschiedenen Konfigurationen, mit verschiedenen Jobs.

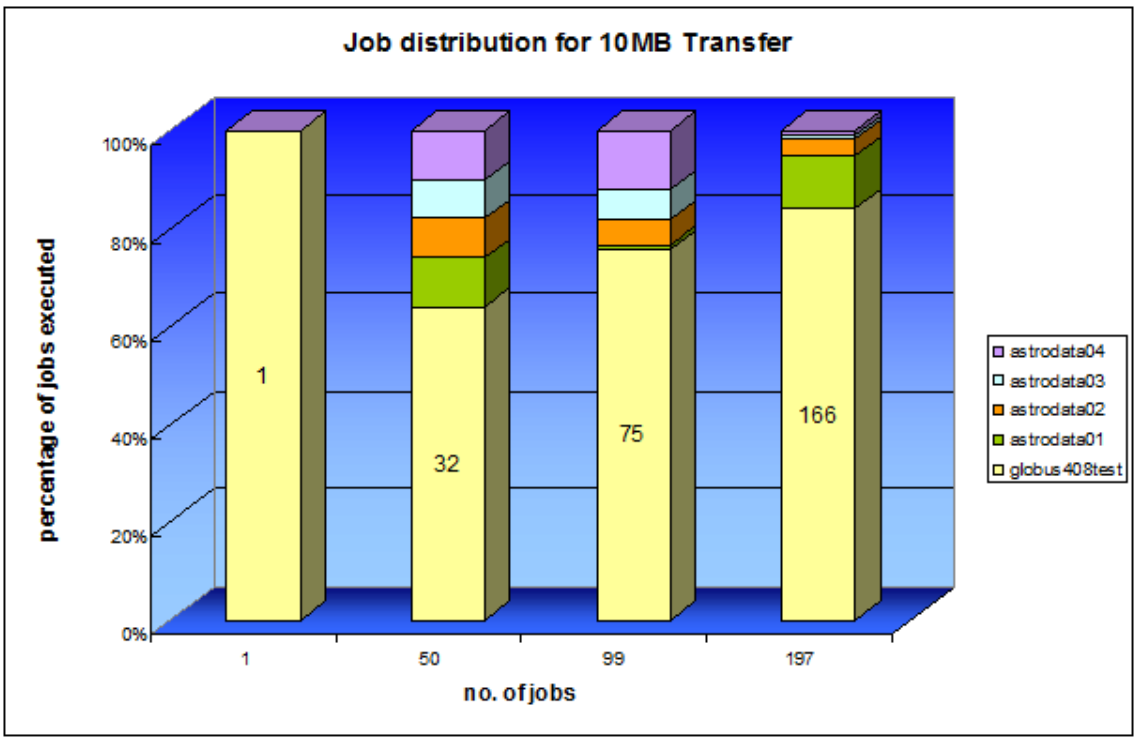
C.2 Scheduling-Testreihe zum Datentransfer



(a) Keine Transferdatei

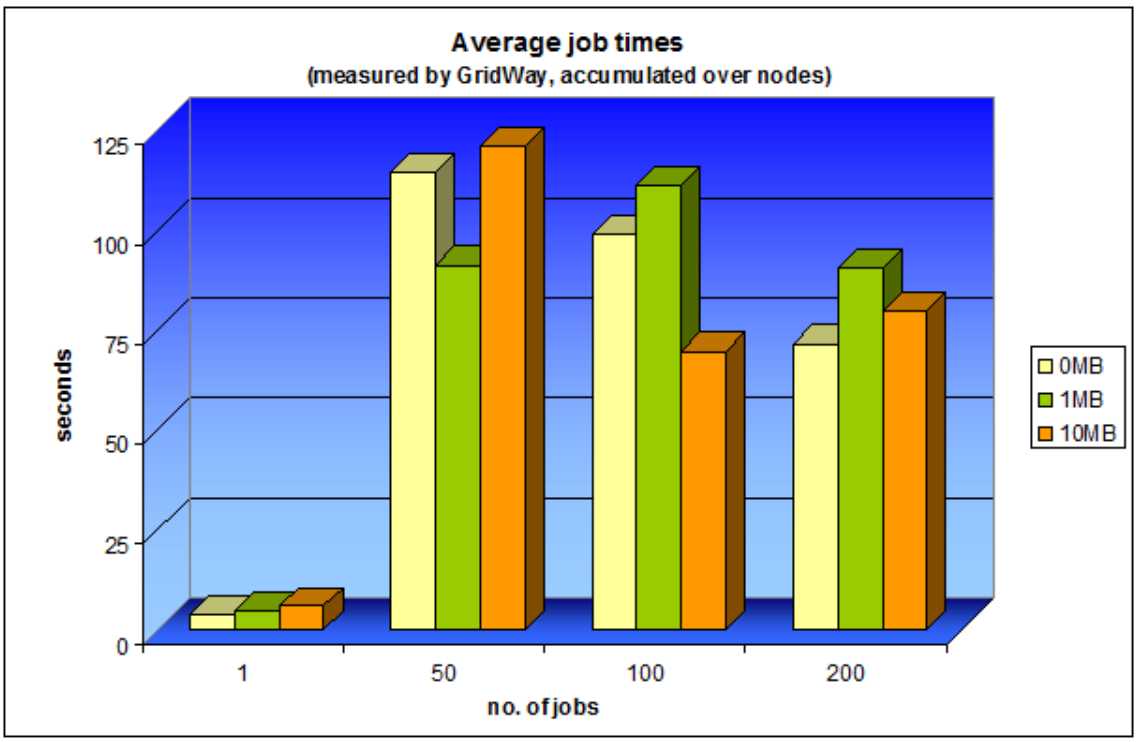


(b) Transferdateigröße 1MB

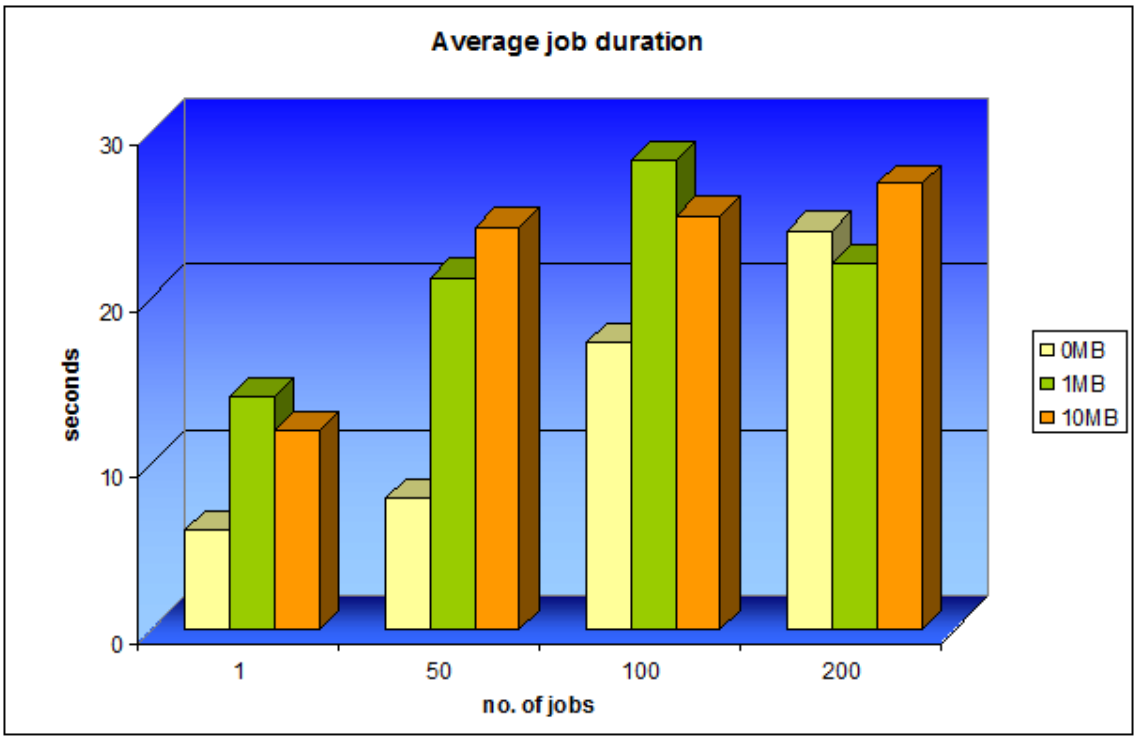


(c) Transferdateigröße 10MB

Abbildung 14: Verteilung der Jobs über die Compute Nodes unter Variation von Transfergröße und Jobanzahl



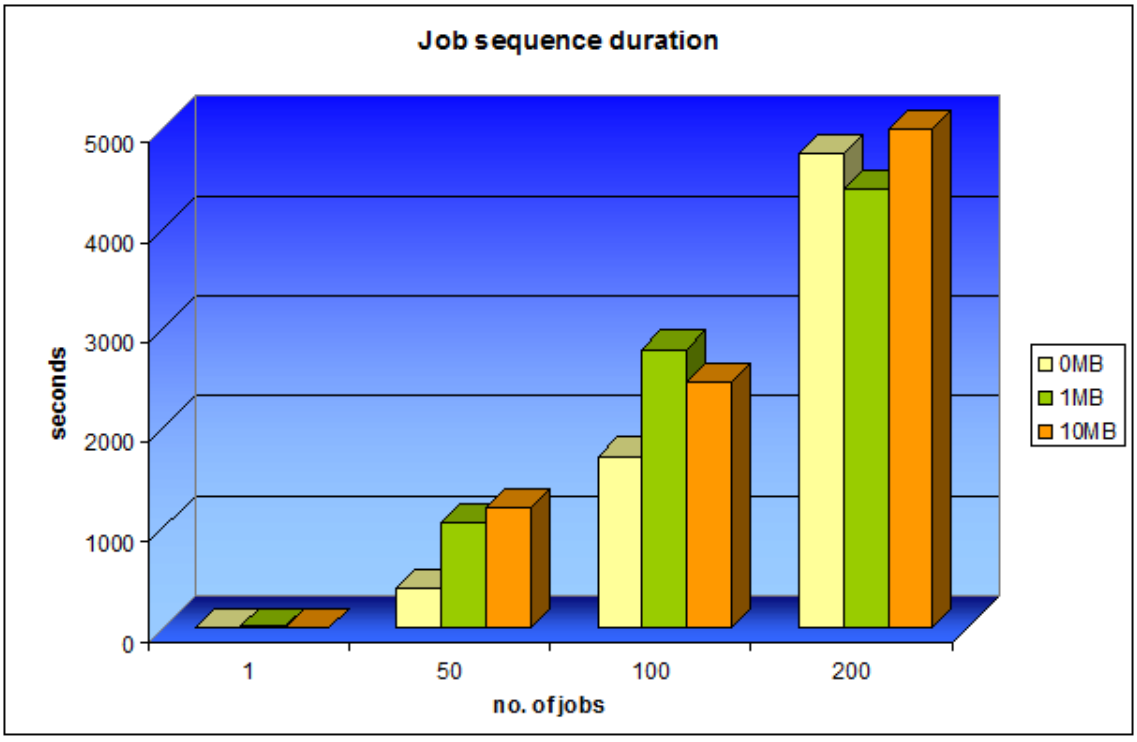
(a) Von GridWay gemessene akkumulierte Jobzeit



(b) Reale Jobdauer

Abbildung 15: Durchschnittliche Jobzeiten in Sekunden über verschiedene Job-Anzahlen und Transferdateigrößen

(a) Von GridWay gemessene akkumulierte Jobzeit



(b) Reale Jobdauer

Abbildung 16: Gesamte Jobsequenzzeiten in Sekunden über verschiedene Job-Anzahlen und Transferdateigrößen

D Prototypische Implementierung

D.1 Job-Submission per Job-Template

In diesem Abschnitt werden im Folgenden einige relevante Zeilen des nachstehenden ausführbaren Shell-Skripts für die Job-Submission per Job-Template erläutert.

Zeile 23: Das iRODS Daten-Objekt mit dem hier absoluten iRODS-Pfad $2 (siehe ARGUMENTS Option im Job-Template, Position 2) wird per iCommand *iget* bezogen und unter dem Namen des Daten-Objekts im job-spezifischen, temporären Globus-Verzeichnis abgelegt.

Zeile 26: Der JHOVE2-Aufruf wird für die mit Aufrufparameter $1 (ARGUMENTS Option, Position 1) benannte Datei abgesetzt. Das hier XML-formatiert JHOVE2-Ergebnis soll in die mit Aufrufparameter $1 benannte Datei geschrieben werden.

Zeile 34: Das JHOVE2-Ergebnis wird in das mit der Variable *$jhove2_results* assoziierte iRODS-Verzeichnis geschrieben. Es ist eigentlich nicht sinnvoll, das Zielverzeichnis im Shell-Skript festzulegen, flexibler wäre die Übergabe als Aufrufparameter.

Zeile 39: Hier wird die PID und DsId des Daten-Objekts zum JHOVE2-Ergebnis, welches zuvor in iRODS gespeichert wurde, per *iquest* (iRODS-Suche im iCAT) ermittelt.

Zeile 53: Wie zuvor, hier wird jedoch die PID und DsId des Original Daten-Objekts ermittelt.

Zeile 68: Setzen der Beziehung zwischen Original und JHOVE2-Ergebnis Daten-Objekt in Fedora. Das Setzen der Beziehung im Rahmen der Schleife ist dem Umstand geschuldet, das im Falle von Multi-Objekten u.U. mehrere Beziehungen gesetzt werden müssen, da ein Daten-Objekt bei diesem Objekt-Typ mit mehreren PID-DsId-Kombinationen verbunden sein kann.

Listing 2: Shell-Skript: jhove2_irods.sh

```
#!/bin/bash

# $0: script file name
# $1: file name of JHOVE-Output
# $2: iRODS-Path of the file to process
# $3: file name, under which the loaded iRODS file ($2)
#     should be saved on globus
# $4: file name for the jhove result

host="localhost"
port=8080
user=fedoraAdmin
password=fedoraAdmin
predicate=wissgrid%3AisMetaOf

source ~/.profile

# create path for Results, if not exist
```

```
jhove2_results=/tempZone/home/rods/s/jhove2_results
imkdir -p $jhove2_results

# get data object
/opt/iRODS/clients/icommands/bin/iget $2

# submit jhove job
jhove2.sh -k -d XML -o $1 $3

# split the iRODS path in collection and file name part
a=$2
collName=${a%/*}
dataName=${a##*/}

# ingest the JHOVE result in iRODS
iput -f $1 $jhove2_results/$4

newDataName=$4

# get pid/dsid of the new datastream (jhove result)
newPidDsIdRelation=`iquest "%s" "SELECT META_DATA_ATTR_VALUE
 WHERE META_DATA_ATTR_NAME = 'PID_DSID_RELATION' AND
 DATA_NAME = '$newDataName' AND COLL_NAME = '$jhove2_results'" `

  newPid=${newPidDsIdRelation%,*}
  newDsid=${newPidDsIdRelation##*,}

  newNamespace=${newPid%:*}
  newId=${newPid##*:}

  subject=info%3Afedora%2f$newNamespace%3A$newId%2f$newDsid


# get pid/dsid of the old datastream (processed iRODS file)
allPidDsIdRelations=`iquest "%s" "SELECT META_DATA_ATTR_VALUE
 WHERE META_DATA_ATTR_NAME = 'PID_DSID_RELATION' AND
 DATA_NAME = '$dataName' AND COLL_NAME = '$collName'" `

for pidDsIdRelation in $allPidDsIdRelations
do
  pid=${pidDsIdRelation%,*}
  dsid=${pidDsIdRelation##*,}

  namespace=${pid%:*}
  id=${pid##*:}

  object=info%3Afedora%2f$namespace%3A$id%2f$dsid
```

```
    # set the relationship in Fedora
    curl --request POST
     "http://$host:$port/fedora/objects/$newNamespace:$newId/
70   relationships/new?subject=$subject&predicate=$predicate&
     object=$object&isLiteral=false" -u $user:$password

done
```

D.2 Job-Submission per API

Hier findet sich der Quellcode der beiden Klassen für die Job-Submission per API, sowie eine kurze Erläuterung der wesentlichen Programmzeilen der Klasse GW_JobSubmission.

Zeile 111: Öffnen einer DRMAA-Session, über welche u.a. das Job-Template erstellt, die Job-Submission abgesetzt oder der Status der Ausführung ermittelt werden kann.

Zeile 119: In der Schleife wird für jede zu verarbeitende Datei ein eigenes Job-Template erzeugt. Diese Job-Templates werden auch in dem Verzeichnis gespeichert, in dem das Java-Programm aufgerufen wird. Standardmäßig werden die Job-Templates jedoch bei erfolgreicher Job-Ausführung wieder gelöscht. Dieses Verhalten kann mit der Aufruf-Option *-l* bzw. *–log* geändert werden, dann bleiben die generierten Job-Templates, und die Log-Dateien *stdout.<job-id>* und *stderr.<job-id>* erhalten.

Zeile 131: Festlegen des Namens des JHOVE2-Outputs, welcher im job-spezifischen Job-Verzeichnis temporär gespeichert wird. Dies entspricht dem ersten Argument der Option *ARGUMENTS*. Die vier Argumente, die zuvor im Job-Template (Listing 1) fest hinterlegt waren, werden hier per API-Funktionen gesetzt. Das gilt auch für das Setzen der *INPUT_FILES*-Option in Zeile 145, das Setzen der Option *REQUIREMENTS* in Zeile 147 oder das Setzen der Option *EXECUTABLE* in Zeile 154.

Zeile 157: Hier erfolgt die Job-Submission.

Zeile 169: Die erzeugten Threads (Klasse JobAgent, Listing 4) warten das Ende der Jobausführung ab und geben eine Statusmeldung aus, ob der Job erfolgreich beendet werden konnte.

Zeile 172: Die für das Job-Template reservierten Ressourcen müssen explizit freigegeben werden.

Zeile 187: Die Session wird geschlossen, danach ist ein Abholen von Statusmeldungen nicht mehr möglich. Da nicht sichergestellt werden kann, dass die Threads unmittelbar nach Beendigung der Job-Ausführung den Prozessor zugeteilt bekommen und den Status abholen, muss der Hauptprozess 10 Sekunden pausieren, damit ruhende Threads die Möglichkeit haben, den Status abzuholen. Wenn ein Thread nicht in der Lage ist, den Status abzuholen, erfolgt eine Ausgabe „Job <jobId> finished with unclear conditions “. In der Regel wird der Job auch dann korrekt abgeschlossen sein; wenn das jedoch wider Erwarten nicht der Fall ist, so kann dies bei einer bereits beendeten Session nur mithilfe der Datei *stderr.<job-id>* (lokal zum Java-Programm abgelegt) festgestellt werden.

Listing 3: Java-Programm: GW_JobSubmission.java

```
package de.unigoettingen.sub.wdf;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.ggf.drmaa.DrmaaException;
import org.ggf.drmaa.GridWayJobTemplate;
import org.ggf.drmaa.GridWaySession;
import org.ggf.drmaa.JobInfo;
import org.ggf.drmaa.JobTemplate;
import org.ggf.drmaa.Session;
import org.ggf.drmaa.SessionFactory;

public class GW_JobSubmission {

  String args[];

  SessionFactory factory;
  Session session;

  Option option1, option2, option3, option4;
  Options options;

  CommandLineParser parser = null;
  CommandLine cmd = null;

  List<String> idList = new ArrayList<String>();

  public static void main(String[] args) {
    new GW_JobSubmission(args);
  }

  public GW_JobSubmission(String args[]) {
    this.args = args;
```

```
      init();
48    process();
    }

    private void init() {
      factory = SessionFactory.getFactory();
53    session = factory.getSession();

      option1 = new Option("f", "comma separated list of files");
      option1.setRequired(false);
      option1.setArgs(Option.UNLIMITED_VALUES);
58    option1.setArgName("filename[,filename,...]");
      option1.setLongOpt("file");

      option2 = new Option(
        "l",
63      "keep stdout.<id>, " +
          "stderr.<id> and generated Job-Templates for " +
            "any processed file");
      option2.setRequired(false);
      option2.setLongOpt("log");
68
      option3 = new Option("h", "print this usage infomation");
      option3.setRequired(false);
      option3.setLongOpt("help");

73    options = new Options();
      options.addOption(option1);
      options.addOption(option2);
      options.addOption(option3);

78    parser = new BasicParser();
      cmd = null;
    }

    public void process() {
83    try {
        cmd = parser.parse(options, args);
      } catch (ParseException e) {
        System.out.println("ERROR: " +
          e.getClass() + ": " +
88          e.getMessage());
        HelpFormatter formatter = new HelpFormatter();
        formatter.printHelp("GW_JobSubmission", options);
        System.exit(-1);
      }
93
      if (cmd.hasOption("h") ||
```

```
      cmd.hasOption("?") || this.args.length == 0) {
      HelpFormatter formatter = new HelpFormatter();
      formatter.printHelp("GW_JobSubmission", options);
98    System.exit(0);
    }

    if (cmd.hasOption("file")) {
      String str = cmd.getOptionValue("file");
103   List<String> files = Arrays.asList(str.split(","));
      processFiles(files);
    }
  }

108 public void processFiles(List<String> files) {

    try {
      session.init(null);
      System.out.println("--- Session Init success --- ");
113   System.out.println("---");

      for (String f : files) {
        String file = f.trim();
        System.out.println("    Processing file: " + file);
118
        GridWayJobTemplate jt = (GridWayJobTemplate) session
          .createJobTemplate();

        int i = file.lastIndexOf("/");
123     String filename = null;
        if (i != -1)
          filename = file.substring(i + 1);
        else
          filename = file;
128
        ArrayList jobArgs = new ArrayList();

        jobArgs.add("out");
        jobArgs.add(file);
133     jobArgs.add(filename);
        jobArgs.add(filename + "_jhove2_out.xml");

        String cwd;
        jt.setWorkingDirectory(
138       java.lang.System.getProperty("user.dir"));
        cwd = jt.getWorkingDirectory();

        String name;
        jt.setJobName(filename + ".jt");
```

```
143           name = jt.getJobName();

              jt.setInputPath("jhove2_irods.sh");

              jt.setRequirements(
148             "HOSTNAME=\"globus408test.sub.uni-goettingen.de\" | " +
                "HOSTNAME=\"astrodata04.gac-grid.org\" | " +
                "HOSTNAME=\"tglobe.sub.uni-goettingen.de\"");
              jt.setRescheduleOnFailure("no");
              jt.setNumberOfRetries("0");
153
              jt.setRemoteCommand("./jhove2_irods.sh");
              jt.setArgs(jobArgs);

              String id = session.runJob(jt);
158           idList.add(id);

              System.out.println("     Job successfully submitted ID: " +
                id);

163           if (!cmd.hasOption("l")) {
                if (new File(cwd + "/" + jt.getJobName()).delete())
                  System.out.println("     Local Job-Template " + cwd
                    + "/" + jt.getJobName() + " deleted");
              }
168
              new Thread(new JobAgent(id, session, cwd,
                cmd.hasOption("l"))).start();

              session.deleteJobTemplate(jt);
173         }

            System.out.println("---");

            session.synchronize(
178           Collections.singletonList(Session.JOB_IDS_SESSION_ALL),
              Session.TIMEOUT_WAIT_FOREVER, false);

            try {
              Thread.sleep(10000);
183         } catch (InterruptedException e) {
              e.printStackTrace();
            }

            session.exit();
188         System.out.println("--- Session Exit success ---");

          } catch (DrmaaException e) {
```

```
        e.printStackTrace();
      }
193   }
    }
```

Listing 4: Java-Programm: JobAgent.java

```
 1  package de.unigoettingen.sub.wdf;

    import java.io.File;
    import java.util.Iterator;
    import java.util.Map;
 6
    import org.ggf.drmaa.DrmaaException;
    import org.ggf.drmaa.JobInfo;
    import org.ggf.drmaa.Session;
    import org.ggf.drmaa.SessionFactory;
11
    public class JobAgent implements Runnable {

      String jobId;
      String cwd;
16    boolean log;

      Session session;

      public JobAgent() {
21
      }

      public JobAgent(String id, Session session,
          String cwd, boolean log) {
26
        this.jobId = id;
        this.cwd = cwd;
        this.log = log;
        this.session = session;
31    }

      @Override
      public void run() {
        JobInfo info;
36      try {
          info = this.session.wait(jobId,
          Session.TIMEOUT_WAIT_FOREVER);

          if (info.wasAborted())
41          System.out.println("Job " + this.jobId + " never ran");
```

```
        else if (info.hasExited()) {
          System.out.println("Job " + this.jobId +
            " finished regularly with exit status "
            + info.getExitStatus());
46
          if (!log) {
            if (new File(cwd + "/stdout." + this.jobId).delete())
              System.out.println("   stdout." + this.jobId +
                " deleted");
51
            if (new File(cwd + "/stderr." + this.jobId).delete())
              System.out.println("   stderr." + this.jobId +
                " deleted");
          }
56      } else if (info.hasSignaled())
          System.out.println("Job " + this.jobId +
            " finished due to signal " +
            info.getTerminatingSignal());
        else
61        System.out.println("Job " + this.jobId +
            " finished with unclear conditions");

      } catch (DrmaaException e) {
        e.printStackTrace();
66    }
    }
  }
```

Literatur

[1] Globus Alliance. Community scheduler framework. http://www.globus.org/grid_software/computation/csf.php, 2009.

[2] Globus Alliance. Globus alliance. http://www.globus.org/, 04 2011.

[3] C3Grid - INAD: Towards an Infrastructure for General Access to Climate Data. https://verc.enes.org/c3web/about-c3grid/c3-inad, 2011.

[4] C3Grid. C3grid - collaborative climate community data and processing grid. http://www.c3grid.de/, 04 2011.

[5] TU Delft. Koala co-allocating grid scheduler. http://www.vl-e.nl, 04 2009.

[6] A. Farquhar and H. Hockx-Yu. Planets: Integrated services for digital preservation. *Int. Journal of Digital Curation*, 2(2):88–99, 2007.

[7] FITS - File Information Tool Set. http://code.google.com/p/fits/, 2011.

[8] The Global Grid Forum. Open grid services infrastructure (ogsi). http://www.ggf.org/documents/GFD.15.pdf, 06 2003.

[9] The Grid Workflow Forum. Generic workflow execution service. http://www.gridworkflow.org/snips/gridworkflow/space/GWES, 04 2011.

[10] GridWay. Gridway scheduler. http://www.gridway.org/, 04 2011.

[11] GridWay User Documentation. http://www.gridway.org/doku.php?id=documentation:release_5.8:ug, 2011.

[12] JHOVE2 - The Next-Generation Architecture for Format-Aware Characterization. http://www.jhove2.org/, 2011.

[13] WissGrid Konsortium. Wissgrid-spezifikation: Grid-repository. http://www.wissgrid.de/publikationen/deliverables/wp3/WissGrid-D3.5.2-grid-repository-spezifikation.pdf, 04 2008.

[14] WissGrid Konsortium. Wissgrid-spezifikation: Langzeitarchivierungsdienste. http://www.wissgrid.de/publikationen/deliverables/wp3/WissGrid-D3.4.2-lza-dienste-spezifikation.pdf, 04 2010.

[15] University of Amsterdam. Virtual laboratory for e-science. http://www.vl-e.nl, 10 2008.

[16] A.D. Peris, J. Hernandez, E. Huedo, and I.M. Llorente. Data location-aware job scheduling in the grid. application to the gridway metascheduler. In *Journal of Physics: Conference Series*, volume 219, page 062043. IOP Publishing, 2010.

[17] O. Sonmez, H.H. Mohamed, and D.H.J. Epema. Communication-aware job placement policies for the koala grid scheduler. In *Proc. of the Second IEEE International Conference on e-Science and Grid Computing*, pages 79–87, 2006.

[18] TextGrid. Textgrid - vernetzte forschungsumgebungen für e-humanities. http://www.textgrid.org/, 04 2011.

[19] WisNetGrid. Semantische beschreibungssprache für web-dienste. http://wisnetgrid.org/index.php?option=com_jdownloads&Itemid=0&view=finish&catid=4&cid=6&lang=de, 06 2010.

[20] WisNetGrid. Wissensnetzwerke im grid. http://wisnetgrid.org/, 12 2010.