Hauptseminar Telematik

WS 2002/03

Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Praktische Informatik und Medieninformatik

Clustern mit Condor

Betreuer:
Dipl. Inf. Thorsten Strufe

Bearbeiter:
Dirk Harms
dirk.harms@in.stud.tu-ilmenau.de

Inhalt:

1. Einleitung

Bei Condor handelt es sich um ein Software-System zur Ausführung von Programmen, welche hohen Rechenaufwand benötigen, auf ungenutzten Workstations. Die Jobs werden dabei in einer Jobqueue verwaltet und automatisch auf die Rechner aufgeteilt.
Condor ist 1988 aus dem Remote Unix Projekt hervorgegangen und wird seit dem an der University of Wisconsin in Madison ständig weiter entwickelt.
Im Vordergrund bei der Nutzung der Rechner zur Ausführung der Jobs steht, dass der Besitzer des Rechners die volle Kontrolle hat. Er kann selbst definieren, unter welchen Bedingungen Jobs auf seinem Rechner ausgeführt werden.
Condor wurde für High Throughput Computing (HTC) entwickelt. Beim HTC ist es nicht wichtig, dass sehr viel Rechenleistung pro Sekunde zu jeder Zeit zur Verfügung steht, sondern dass über einen längeren Zeitraum wie eine Woche oder einen Monat möglichst viel Rechenleistung zur Verfügung steht. Damit ist es zum Beispiel für Simulationen geeignet, die sehr lange auf eventuell sehr vielen verschiedenen Eingaben durchgeführt werden müssen. Ein Job wird bei Condor im allgemeinen auf einem Rechner ausgeführt, was natürlich die verfügbare Leistung begrenzt. Mittels der PVM Erweiterung können allerdings auch parallele Programme auf mehreren Rechnern ausgeführt werden.
Sämtliche Informationen über Jobs und Ressourcen im Condor Pool werden in sogenannten ClassAd's gespeichert. Ein ClassAd ist eine Sammlung von eindeutigen Ausdrücken. Dabei gibt es keine Vorschriften, wie viele oder welche Ausdrücke genau der ClassAd enthalten muss, was die ClassAd's sehr flexibel und leicht erweiterbar macht.
Die Jobs werden in Condor in bestimmte Arten von Programmen aufgeteilt, die Universes genannt werden. Die unterschiedlichen Universes stellen verschieden Anforderungen an den Job, und das Universe eines Jobs bestimmt auch, welche Features von Condor nutzbar sind.
Condor ist als Installationspaket für Linux, Windows, Solaris, IRIX, HPUX und Digital Unix Systeme frei verfügbar. Die Quelltexte sind allerdings nicht verfügbar.

2. Struktur eines Condor Clusters

2.1 Die Rechnertypen

In jedem Condor Pool gibt es genau einen Central Manager, der die Koordination des Pools übernimmt. Er sammelt Informationen über alle im Pool verfügbaren Rechner und alle in Auftrag gegeben Jobs. Mit Hilfe dieser Informationen teilt er die Jobs den freien Rechnern zu. Auf dem Central Manager laufen als Daemons der Master, der Collector und der Negotiator.
Weiterhin gibt es im Pool Submit Rechner, von welchen aus Jobs in Auftrag geben werden können. Auf diesen laufen als Daemons der Master und der Schedd. Bei der Ausführung eines Jobs wird für diesen Job noch der Shadow gestartet.
Außerdem gibt es Execute Rechner, auf denen die Jobs ausgeführt werden. Auf diesen Rechnern müssen der Master und der Startd laufen. Für das Ausführen eines Jobs wird hier noch der Starter gestartet.
Die Rechnertypen können natürlich auch kombiniert werden, sodass Rechner gleichzeitig Submit und Execute Rechner sein können. Auch der Central Manager kann mit Submit und Execute Rechner sein.

2.2 Die Condor Daemons

Auf jedem Rechner läuft der Master, dessen Aufgabe es ist, die anderen Daemons auf dem Rechner zu starten und ihre Ausführung zu überwachen. Sollte ein Daemon nicht mehr laufen oder beim Starten abstürzen, wird er vom Master automatisch neu gestartet. Dabei werden die Zeitintervalle zwischen den Neustarts immer weiter erhöht, um eine zu starke Belastung des Rechners zu verhindern, falls es sich um einen fehlerhaften Daemon handelt, der nie ordentlich startet. Außerdem überwacht der Master, ob für Daemons neue Binarys installiert wurden, und startet die Daemons gegebenenfalls mit den neuen Binarys neu.

Der Collector läuft nur auf dem Central Manager. Bei ihm melden sich alle Rechner an, so dass er immer die IP's und den Port der Daemons von allen Rechnern im Condor Pool kennt und diese den Rechnern, die mit andern Rechnern im Pool kommunizieren wollen, mitteilen kann. Der Collector bekommt von allen Execute Rechnern regelmäßig ihren Status mitgeteilt. Dazu zählen allgemeine Informationen über den Rechner wie Architektur, Betriebssystem, Rechenleistung und Ram Speicher aber auch der momentane Zustand des Rechners wie Auslastung, Zeit, die er bereits idle ist, und unter welchen Bedingungen er Jobs ausführt. Von den Submit Rechnen bekommt der Collector die Aufträge zugesandt, wenn sie auf dem Rechner in Auftrag gegeben werden.

Der Negotiator läuft ebenfalls nur auf dem Central Manager. Er holt sich vom Collector den Status aller Rechner und alle Jobs. Mit diesen Informationen versucht er, die Jobs gemäss ihren Anforderungen und der Userprioritäten auf die freien Rechner zu verteilen. Wird ein Job einem Rechner zugeordnet, meldet dies der Negotiator dem Schedd des Rechners, auf dem er in Auftrag gegeben wurde, und dem Startd des Rechners, wo er ausgeführt werden soll.

Der Schedd muss auf allen Rechnern laufen, von dem aus User Aufträge für den Condor Cluster geben wollen. Er nimmt die Aufträge der User entgegen, speichert sie in der lokalen Job Queue, und meldet sie an den Negotiator des Central Manager weiter. Wird dem Schedd vom Negotiator mitgeteilt, dass einer seiner Jobs ausgeführt wird, startet er den Shadow, welcher die Systemaufrufe des Jobs lokal ausführt und das Ergebnis an den Starter des Execute Rechners schickt, damit dieser es an den Job weiterleitet.

Auf allen Rechnern, die Jobs ausführen sollen, muss der Startd laufen. Dieser überwacht, ob der Rechnern idle ist bzw. die Bedingungen zur Ausführung von Jobs erfüllt sind. Er meldet regelmäßig den Status des Rechners an den Collector weiter. Sollte der Rechner vom Negotiator einen Job zugewiesen bekommen, startet er für die Ausführung des Jobs den Starter. Dieser leitet alle Dateisystem-Operationen und allen Output des Jobs an den Shadow auf dem Rechner, von dem der Job in Auftrag gegeben wurde, weiter.

Auf Execute Rechnern, die unter IRIX oder Digital Unix laufen, muss außerdem noch der kdb laufen. Dieser überwacht auf diesen Systemen die Maus und Keyboardaktivität und meldet diese an den Startd.

3. Features von Condor

3.1 Jobmanagement

Condor stellt ein Job Management zur Verfügung, mit dem es möglich ist, eine große Anzahl von sich ähnelnden Jobs auf einfache Weise gleichzeitig in Auftrag zu geben. Es kümmert sich automatisch darum, dass Jobs auf anderen Rechnern neu gestartet oder fortgesetzt werden, sollten die Rechner, auf denen sie ausgeführt werden, verschwinden oder die Jobausführung abbrechen.
Das Jobmanagement von Condor ermöglicht es dem User, genaue Anforderungen an den Rechner zu stellen, auf dem der Job ausgeführt werden soll. Damit kann der User zum Beispiel sicherstellen, dass sein Job innerhalb seiner Dateisystem-Domäne ausgeführt wird oder der Rechner über genügend RAM und Festplattenspeicher verfügt, um den Job auszuführen.
Der User kann zusätzlich dazu definieren, nach welchen Gesichtspunkten die Rechner für seinen Job bewertet werden sollen. Damit kann erreicht werden, dass Jobs nicht auf für sie ungeeigneten Rechnern laufen, während besser geeignete Rechner idle sind.
Des weiteren kümmert sich das Jobmanagement automatisch um die Anpassung der Userprioritäten, so dass User, die bereits viele Ressourcen belegen oder kürzlich viele Ressourcen in Anspruch genommen haben, eine niedrigere Priorität bekommen. Sollte ein User eine zu niedrige Priorität haben, können auch laufenden Jobs von ihm beendet werden, um Jobs von Usern mit deutlich höherer Priorität die Ressourcen zuzuteilen. Damit ist sichergestellt, dass nicht alle User warten müssen, weil ein User bereits alle Ressourcen mit sehr lange dauernden Jobs belegt.

3.2 Umleitung von Input und Output

Wenn Jobs ausgeführt werden, wird ihre Ausgabe, die normalerweise nach stdout gehen würde, in ein File umgeleitet. Ebenso wird die Fehlerausgabe stderr in ein File umgeleitet.
Außerdem kann der Input von einem File an den Job geleitet werden. Dabei ist allerdings kein interaktiver Input möglich.

3.3 Checkpointing

Checkpoints speichern den Zustand des laufenden Jobs. Sie enthalten neben einem kompletten Speicherabbild des Jobs auch Informationen darüber, welche Dateien gerade geöffnet sind und die momentane Lese- bzw. Schreibposition in diesen Dateien.
Diese Checkpoints werden während der Abarbeitung eines Jobs regelmäßig angelegt. Sollte nun ein Rechner, welcher gerade einen Job ausführt, aus dem Pool verschwinden, weil er zum Beispiel heruntergefahren wird oder abgestürzt ist, kann der Job an dem letzten Checkpoint auf einem anderen Rechner fortgesetzt werden.
Wenn ein Nutzer an seinen Rechner zurückkehrt, wird ebenfalls ein Checkpoint angelegt, bevor die Jobausführung auf dem Rechner beendet wird.
Durch das Checkpointing wird somit sichergestellt, dass nie die ganze bereits in eine Job investierte Rechenzeit verloren geht, sondern nur der relativ geringe Teil seit dem letzten Checkpoint.

Damit kann erreicht werden, dass auch sehr lange Jobs, die Monate bis zur Fertigstellung benötigen, problemlos auf normalen Workstations ausgeführt werden können und keinen extra Rechner benötigen, der die ganze Zeit der Ausführung ununterbrochen zur Verfügung steht.
Im Normalfall bemerkt der Nutzer am Output des Jobs nicht, ob ein Job während der Ausführung einmal unterbrochen und später an einem Checkpoint fortgesetzt wurde. Sämtliche geöffneten Dateien und auch der umgeleitete Output wird beim Fortsetzen auf den Zustand beim Anlegen des Checkpoints zurückgesetzt.
Ich konnte allerdings beim Test mit dem Raytracer Programm Povray feststellen, dass das Ausgabebild nicht zurückgesetzt wurde und somit fehlerhaft war. Dies war zu beobachten, wenn man die Ausführung auf einem Rechner gekillt hat, so dass an einem früheren Checkpoint fortgesetzt werden musste. Beim sanften Beenden des Jobs mit einem Checkpoint, wie es bei der Rückkehr eines Users stattfindet, war nach dem Fortsetzen an diesem Checkpoint kein Fehler feststellbar. Der Fehler lag also an dem Zurückrollen, wenn bereits über den Checkpoint hinaus weitergearbeitet worden war.
Daraufhin schrieb ich ein kurzes Testprogramm, um genauer festzustellen, wann es zu solchen Fehlern im Output kommt. Der erste Test schrieb entweder kontinuierlich auf stdout oder aber in eine Datei, die beim Start des Programms zum Schreiben geöffnet und erst kurz vor dem Beenden wieder geschlossen wurde. Dabei konnte ich keine Fehler feststellen. Der Output sowohl des umgeleiteten stdout als auch in der Datei wurden immer korrekt zurückgesetzt. Auch ein Öffnen der Datei zum Lesen und Schreiben, was laut Anleitung zu Problemen führen kann, lieferte eine korrekten Output. Nun modifizierte ich das Programm so, dass die Outputdatei vor jedem Schreiben zum Anhängen geöffnet und nach dem Schreiben sofort wieder geschlossen wurde. Dies führte bei einem Abbruch der Ausführung nach dem Fortsetzen am Checkpoint, genau wie beim Povray beobachtet, dazu, dass ein Teil des Output doppelt vorhanden war. Wenn man Programme entwickelt, die später in einem Condor Cluster ausgeführt werden sollen, sollte man also darauf achten, alle Ausgabedateien am Anfang zu öffnen und während der Ausführung geöffnet zu lassen.

3.4 Remote System Calls

Die Remote System Calls von Condor sorgen dafür, dass Zugriffe eines Jobs auf das Dateisystem abgefangen und als Anforderung an den Shadow auf dem Rechner, von dem der Job in Auftrag gegeben wurde, gesandt werden. Dort werden diese dann ausgeführt und das Ergebnis an den ausführenden Rechner zurückgeschickt. Da der Shadow möglichst auch mit den Rechten des Users, der den Job in Auftrag gegeben hat, gestartet wird, findet der Job unabhängig davon, wo er ausgeführt wird, genau die Dateisystem Umgebung wieder, die er hätte, wenn ihn der User lokal gestartet hätte.

3.5 File Transfer

File Transfers sind für Jobs gedacht, wo Remote System Calls nicht möglich sind, aber bestimmte Dateien für die Jobausführung benötigt oder Ausgabedateien erstellt werden. In solchen Fällen kann der User, wenn er den Job in Auftrag gibt, angeben, welche Dateien für die Jobausführung mit zum Execute Rechner transferiert werden sollen, und welche Dateien nach dem Ausführen des Jobs auf seinen Rechner übertragen werden sollen. Dadurch ist sichergestellt, dass man nicht auf ein gemeinsames Dateisystem auf den Rechnern des Pools angewiesen ist.

4. Installation von Condor

4.1 Vorüberlegungen

Als erstes sollte man sich überlegen, welcher Rechner der Central Manager werden soll. Dieser sollte dauerhaft laufen und über eine verhältnismäßig gute Netzanbindung verfügen. Auf diesem sollte Condor auch zuerst installiert werden.
Wenn alle Rechner des Pools über ein gemeinsames Dateisystem verfügen, ist es günstig, Condor in das gemeinsame Dateisystem zu installieren, da die Condorinstallation mit 100-200MB je nach Architektur nicht gerade klein ist. Außerdem vereinfacht es die Konfiguration, wenn alle Rechner auf eine gemeinsame Konfigurationsdatei zugreifen können.
Dann sollte man sich überlegen, als welcher User Condor installiert wird. Es wird empfohlen, einen User Condor anzulegen, was allerdings nicht notwendig ist, sondern es einem lediglich die Definition einiger Umgebungsvariablen erspart.
Des weiteren sollte man sich überlegen, mit welchen Rechten Condor starten soll. Man muss Condor nicht mit root Rechten starten. Allerdings ergeben sich einige Nachteile, wenn Condor nur normale Userrechte hat. Der größte Nachteil, der sich ergibt, ist das Entstehen eines Sicherheitslochs. So ist Condor, wenn es nur mit normalen User Rechten läuft, nicht in der Lage, die Jobs als der auftraggebende User oder als User Nobody auszuführen, sondern muss sie als der User ausführen, mit dessen Rechten es gestartet wurde. Damit könnte ein User einen Job in Auftrag geben, der als erstes die Condor Daemons killt, und dann mit dem eigentlichen Rechnen beginnt. Dieser Job würde dann ständig laufen und könnte nicht von Condor beendet werden, wenn der User an den Rechner zurückkehrt oder User mit höherer Priorität Ressourcen benötigen. Da es für Condor so aussieht, als wäre der Rechner abgestürzt, würde der Job dann auf dem nächsten Rechner gestartet, wo er dasselbe tun kann. Damit könnte ein User mit der Zeit den kompletten Pool für sich allein nutzen, ohne dass seine Priorität sinkt und er für Rechnerbesitzer oder andere User irgendwelche Ressourcen freigeben muss. Ein weiterer Nachteil, wenn Condor nicht als root läuft, ist, dass die User dafür sorgen müssen, dass der Condor-User Zugriffsrechte auf alle für den Job benötigten Dateien erhält. Das führt dazu, dass sie auch die Dateien für den Output und eventuelles Logging vor dem in Auftrag geben des Jobs anlegen müssen, da diese beim automatischen Anlegen beim Aufruf von condor_submit mit den falschen Rechten angelegt werden, und der Job während der Ausführung nicht auf diese zugreifen kann.
Es empfiehlt sich also, vor allem wenn eine größer Anzahl von Usern auf den Pool Zugriff hat, Condor mit root Rechten laufen zu lassen.
Als letztes sollte man sich überlegen, wo die rechnerspezifischen Verzeichnisse liegen, falls man Condor in einem gemeinsamen Dateisystem installiert. Condor schlägt dabei vor, im Installationsverzeichnis ein Hosts Verzeichnis anzulegen, in welchem für jeden Rechner des Pools ein eigenes Verzeichnis angelegt wird. Damit würden auch die rechnerspezifischen Verzeichnisse mit im gemeinsamen Dateisystem liegen. Dies macht für das Konfigurationsfile und das Log Verzeichnis auch durchaus Sinn, da es die Administration und Überwachung des Pools erleichtert. Wenn man nur über ein recht langsames gemeinsames Dateisystem oder wenig Platz verfügt, macht es allerdings Sinn, das Execute Verzeichnis, in das die Programme für die Ausführung kopiert werden und in dem die Programme eventuell auch während der Ausführung temporäre Dateien anlegen, auf ein lokales Verzeichnis zu legen. Dasselbe gilt für das Spool Verzeichnis, in dem die für den Job notwendigen Dateien nach dem in Auftrag geben bis zur Ausführung zwischengespeichert werden.

Das Lock Verzeichnis sollte unbedingt auf einer lokalen Platte abgelegt werden. Sowohl mit SMB als auch mit NFS als Dateisystem startete Condor nicht ordentlich, wenn das Verzeichnis nicht lokal lag.
Bei der Verwendung von SMB als Dateisystem gab es auch Probleme, wenn das Execute und das Log Verzeichnis nicht lokal lagen. Wenn man SMB verwendet, sollte man also am besten das ganze lokale Verzeichnis wirklich lokal anlegen und maximal die lokalen Konfigurationsdateien im gemeinsamen Dateisystem ablegen.

4.2 Installation der Rechner des Pools

Zuerst muss der Central Manager installiert werden. Nachdem man Condor heruntergeladen und entpackt hat, geschieht die Installation mittels des mitgelieferten Installationsscripts condor_install. Das Script fragt dann als Erstes nach der Installationsart, wobei „Full“ gewählt werden sollte. Danach wird abgefragt, welche Rechner im Pool sein sollen, wobei sich auch später noch problemlos weitere Rechner in den Pool integrieren lassen. Dann wird noch nach den Installationsverzeichnissen gefragt, welcher Rechner der Centrale Manager wird und Einiges zur Umgebung wie Dateisystemdomäne und User-Domäne. Wenn das Script auf dem Central Manager fertig ist, können die einzelnen Rechner des Pools installiert werden. Sollte Condor in ein gemeinsames Dateisystem installiert worden sein, reicht ein Aufruf von condor_init auf jedem der Rechner, die bereits vorher eingetragen wurden. Dies legt die Verzeichnisse an die auf der lokalen Platte liegen sollen wie zum Beispiel das Lock dir. Sollte Condor nicht in einem gemeinsamen Dateisystem liegen, muss auf jedem der Rechner eine Installation mittels condor_install durchgeführt werden.

4.3 Grundkonfiguration

Nach der Installation mittels des condor_install Skripts ist die Konfiguration bereits auf die eigenen Verzeichnisse angepasst. Es existiert bereits eine vorbereitete lokale Konfigurationsdatei, welche nur noch in das lokale Verzeichnis des Central Manager kopiert werden muss. Änderungen müssen hier im Moment nicht vorgenommen werden. Wichtig ist in dieser Datei erst einmal nur, dass die Liste der zu startenden Daemons hier auch den Collector und der Negotiator enthält, was bei der vorgegeben Datei bereits der Fall ist.
Danach müssen für alle einzelnen Hosts die lokalen Konfigurationsdateien angelegt werden. Diese müssen unter <Lokaldir>/condor_config.lokal von Condor zu finden sein. Anfangs kann diese leer gelassen werden, da die Standardwerte bereits in der gemeinsamen Konfigurationsdatei festgelegt sind. In der gemeinsamen Konfigurationsdatei ist auch festgelegt, dass jeder Host den Master, den Schedd und den Startd startet. Damit könnte jeder Rechner Jobs in Auftrag geben und Jobs ausführen. Soll ein bestimmter Rechner nur Jobs ausführen muss in seiner Konfigurationsdatei eine Zeile „DAEMON_LIST = MASTER, STARTD“ eingefügt werden, um die Einstellung aus der gemeinsamen Konfiguration zu überschreiben.
Sollte man einen User Condor haben und Condor den Vorschlägen gemäss in dessen Homeverzeichnis installiert haben, kann man Condor nun durch Aufruf von condor_master auf den Rechnern starten. Hat man keinen User Condor, müssen noch zwei Umgebungsvariablen gesetzt werden. Die erste ist CONDOR_CONFIG und muss die Position der Konfigurationsdatei enthalten. Alternativ dazu kann man die Konfigurationsdatei auch unter /etc/condor/condor_config ablegen. Dies hat den Vorteil, dass nicht jeder User, der Condor nutzen möchte, die Variable anlegen muss.

Die zweite Umgebungsvariable ist CONDOR_IDS. Sie muss durch einen Punkt getrennt die Userid und die Groupid des Users enthalten, mit dessen Rechten Condor die Logfiles anlegen soll, also im Allgemeinen des Users, welcher die Administration von Condor übernimmt.
Überträgt man die Konfiguration von Condor einem anderen User, ist es möglich, eine Datei /etc/condor/condor_config.root anzulegen. Diese Konfigurationsdatei wird dann als erstes eingelesen, und alle hier gemachten Definitionen können in anderen Konfigurationsdateien nicht überschrieben werden. Es ist somit also möglich, nicht die komplette Konfiguration aus der Hand zu geben, sondern zum Beispiel die Sicherheitseinstellungen selber vorzunehmen.
Wenn man sich in einem frei zugänglichen Netzwerk befindet, ist es sicher auch sinnvoll bevor man Condor startet, in der Konfiguration mit Hilfe der Hostallow/Hostdeny Regeln den Zugriff auf den Pool auf die wirklich zum Pool gehörenden Rechner einzugrenzen, oder gleich mit den Allow/Deny Regeln auch die User einzuschränken, die Zugriff haben.
In der normalen Konfiguration sendet der Central Manager auch wöchentlich eine kurze Statistik an die Entwickler. Wünscht man dies nicht, sollte man die Einträge CONDOR_DEVELOPER und CONDOR_DEVELOPER_COLLECTOR auf NONE setzen.

5. Jobs in Auftrag geben

5.1 Submit-Files für Jobs

Jobs, die in Auftrag gegeben werden sollen, müssen in einem Submit-File beschrieben werden. Diese können zum Beispiel wie folgt aussehen:

```
Executable = ctest
Arguments = 500 100
Universe = standard
Output = ctest.output
Error = ctest.error
Input = ctest.input
Initialdir = test1

Queue
```

`Executabel` gibt dabei das Programm an, was ausgeführt werden soll, `Arguments` die beim Start zu übergebenden Parameter. `Universe` definiert das Condor Universe, in dem der Job ausgeführt werden soll. Die Universes sind gewisse Klassifizierungen von Jobs, die in Kapitel 8 noch genauer beschrieben werden. `Output` und `Error` definieren die Files, in die stdout und stderr umgeleitet werden soll. `Input` gibt das File an, dessen Inhalt als stdin an das Programm geleitet werden soll. Durch `Initialdir` definiert man das Arbeitsverzeichnis des Jobs.
Beendet wird die Definition immer durch `Queue` als Keyword, um den Job zur Jobqueue hinzuzufügen.
Sollen Files für die Jobausführung mit übertragen werden, muss mittels `transfer_input_files` und `transfer_output_files` die Liste der zu übertragenden Dateien angegeben werden. Außerdem muss `transfer_files` als `ONEXIT` oder `ALWAYS` definiert werden. ONEXIT heißt dabei, dass die Output Files nur nach der kompletten Abarbeitung des Jobs übertragen werden, während bei ALWAYS diese auch übertragen werden, wenn die Abarbeitung abgebrochen wird.
Weiterhin kann über `priority` eine Priorität zwischen –20 und +20 für den Job festgelegt werden. Ist diese nicht definiert, wird sie auf Null gesetzt. Diese Priorität ist aber nur zur Unterscheidung der Jobs eines Users interessant. Bei der Entscheidung, wessen Jobs zuerst ausgeführt werden, ist sie irrelevant.
Wenn ein Job nur ausgeführt werden soll, wenn kein anderer Job, also auch kein Job eines anderen Users mehr auszuführen ist, lässt sich dies durch Definition von `nice_user = TRUE` im Submit-File festlegen.

5.2 Requirements

Im Submit-File kann weiterhin ein Requirements-Ausdruck definiert werden. Mit diesem Ausdruck können bestimmte Anforderungen eines Jobs an den Rechner, auf dem er ausgeführt wird, definiert werden. Dabei kann auf alle Attribute des Maschine ClassAd's zugegriffen werden. Dies sind zum Beispiel die Architektur des Rechners, das Betriebssystem, die Ramgrösse, der freie Festplattenspeicher oder die Dateisystemdomäne, welcher der Rechner angehört.
Ein Requirements-Ausdruck, der festlegt, dass zur Ausführung mindestens 64MB Ram vorhanden sein müssen und der Rechner sich in der Domäne prakinf.tu-ilmenau.de befinden muss, sieht dann wie folgt aus:

```
Requirements = ( FileSystemDomain =="prakinf.tu-ilmenau.de")
                    && ( Memory >= 64 )
```

Da die Jobs bei Condor nicht in einer virtuellen Maschine laufen und somit an die Architektur und das Betriebsystem gebunden sind, für das sie kompiliert wurden, werden, wenn Architektur und Betriebsystem nicht definiert sind, automatisch Architektur und Betriebsystem des Rechners, auf dem der Job in Auftrag gegeben wird, zu den Requirements hinzugefügt.

5.3 Rank

Über Rank kann im Submit-File ein Ausdruck angegeben werden, über den Rechner für den Job bewertet werden. Dieser kann wie Requirements auf alle Attribute des Rechners zugreifen. Stehen nun für die Ausführung eines Jobs mehrere Rechner zur Verfügung, wird für jeden der Rank-Ausdruck ausgewertet und der Job auf dem mit dem höchsten Ergebnis ausgeführt. Dies stellt sicher, dass Jobs nicht auf einem langsamen Rechner ausgeführt werden, während ein wesentlich schnellerer Rechner nicht verwendet wird. Dies wäre zwar auch mit einer generellen Bewertung der Rechner möglich, aber über den Rank-Ausdruck hat der User die Möglichkeit, genauer zu definieren, was für die Ausführung seines Jobs wichtiger ist.
Ein Rank-Ausdruck, für einen Job für den eine hohe Floating Point Performance und möglichst viel Ram Speicher notwendig ist, würde dann so aussehen:

```
Rank = (memory * 1000) + kflops
```

Ist vom User kein Rank definiert, werden in der Standardkonfiguration alle Rechner gleich bewertet. Dies ist vor allem dann ungünstig, wenn man Rechner mit sehr unterschiedlicher Leistung im Pool hat und meist mehr Rechner zur Verfügung hat, als Jobs anstehen. Um auch dann ein sinnvolles Ranking zu ermöglichen, kann in der Konfigurationsdatei ein Standard-Rank-Ausdruck definiert werden, der für alle Jobs verwendet wird, bei denen der User keinen eigenen definiert hat. Die Syntax dieses default Rank entspricht der des Rank-Ausdrucks im Submit-File.

5.4 Mehrfachausführung

Es ist auch möglich, gleich mehrere Jobs mit einem Submit-File in Auftrag zu geben. Grundsätzlich gibt es zwei verschiedene Arten, dies zu tun.
Die erste eignet sich vor allem, wenn man eine geringe Menge ähnlicher Jobs ausführen möchte. Dabei wird der erste Job ganz normal im Submit-File definiert und die Definition durch Queue abgeschlossen. Danach werden die für den zweiten Job abweichenden Attribute neu definiert und dieser durch Queue abgeschlossen. Für jeden so in Auftrag gegeben Job wird dann immer die letzte Definition eines Attributes vor dem Queue verwendet.
Dies könnte zum Beispiel wie folgt aussehen:

```
Executable = ctest
Arguments = 500 100
Output = ctest.output
Input = ctest.input
Initialdir = test1

Queue

Arguments = 700 200
Initialdir = test2
Queue
```

Dies eignet sich natürlich nur für eine geringe Anzahl von Jobs und lässt sich für Jobs, die mehrere hundert mal auf verschiedenen Eingaben ausgeführt werden müssen, nur schlecht verwenden.
Um dies zu machen, kann man mittel `Queue n` einen Job n mal in Auftrag geben. In den Attributen kann man dann das Macro $(PROCESS) verwenden welches immer nach dem aktuellen Durchlauf aufgelöst wird. Ein so definierter Job sähe dann zum Beispiel so aus:

```
Executable = ctest
Arguments = 500 100
Output = ctest.output
Input = ctest.input
Initialdir = test$(PROCESS)

Queue 100
```

Dabei würde der Job 100 mal ausgeführt, wobei als Arbeitsverzeichnisse jeweils die Verzeichnisse test0 bis test99 verwendet werden würden.

5.5 Ausführung auf mehren Architekturen

Wie bereits vorher erwähnt, werden die Jobs nicht in einer virtuellen Maschine ausgeführt und sind somit nur auf der Architektur ausführbar, für die sie erstellt wurden. Nun kann es aber sein, dass man ein Programm für mehrere Architekturen und auch Rechner dieser verschiedenen Architekturen im Pool hat und möglichst alle ausnutzen möchte. Um dies zu ermöglichen, muss man im Submit-File in den Requirements alle Architekturen, für die man Binarys hat, erlauben. Das zu verwendende Binary muss dann mittels der Macros $$(Arch) und $$(Opsys) definiert werden, welche nach der Architektur bzw. dem Betriebssystem des Rechners aufgelöst werden, der den Job zugewiesen bekommt. Dies sieht im Submit File dann wie folgt aus:

```
Executable = ctest.$$(Arch).$$(Opsys)
Arguments = 500 100
Output = ctest.outbut
Input = ctest.input
Requirements = ((Arch=="INTEL")&&(OpSys=="LINUX")) ||
               ((Arch=="SUN4u")&&(OpSys=="SOLARIS28"))

Queue
```

Bei der Ausführung würde als Binary dann je nach System ctest.INTEL.LINUX oder ctest.SUN4u.SOLARIS verwendet.

6. Ausführung der Jobs

6.1 Ressourcen im Condor Pool

Die Ressourcen des Condor Pools für die Jobausführung sind alle Rechner des Pools, auf denen ein Startd läuft. Diese melden sich beim Collector an und schicken ihren Maschine ClassAd an diesen. Der Maschine ClassAd enthält alle Informationen über den Rechner, die für einen Job interessant sein könnten, also Hardware, Betriebsystem, freier Festplattenspeicher, Rechenleistung und in welcher Dateisystem-Domäne der Rechner steht. Außerdem steht im ClassAd der momentane Zustand des Rechners, also ob er gerade vom Besitzer verwendet wird, idle ist oder einen Condor Job ausführt. Weiterhin steht der vom Besitzer des Rechners festgelegte Start und Rank-Ausdruck im ClassAd. Mit diesen Ausdrücken definiert der Besitzer des Rechners die Bedingungen, unter denen Jobs auf seinem Rechner ausgeführt werden dürfen.
Der Start-Ausdruck definiert die Bedingungen, unter denen der Rechner für Jobs zur Verfügung steht und kann sowohl auf Attribute des lokalen Maschine ClassAd's zugreifen als auch auf Attribute des Jobs, der dem Rechner zugewiesen werden soll. Zuerst wird der Ausdruck nur lokal ausgewertet, also nur mit den Attributen des Maschine ClassAd's . Ergibt diese lokale Auswertung TRUE, steht der Rechner für Jobs zur Verfügung. Soll dem Rechner nun ein Job zugewiesen werden, wird der Start-Ausdruck sowohl mit den Maschine ClassAd als auch mit den Job ClassAd Attributen ausgewertet. Ergibt diese Auswertung wiederum TRUE, kann der Job auf dem Rechner ausgeführt werden.
Der Start-Ausdruck in der mitgelieferten Konfiguration wird TRUE, wenn der Rechner 15 Minuten nicht verwendet wurde, also 15 Minuten lang keine Maus- und keine Tastaturaktivität vorhanden war, und die load Average auf dem Rechner unter 0.3 liegt.
Da der Start-Ausdruck vom Besitzer des Rechners beliebig umdefiniert werden kann, ist es ihm auch möglich, zu definieren, dass Jobs nur in bestimmten Zeiträumen wie zum Beispiel nach Feierabend oder am Wochenende ausgeführt werden dürfen. Dadurch, dass der Rechnerbesitzer dabei auch auf die Attribute des Jobs zugreifen kann, kann er auch verhindern, dass Jobs bestimmter Nutzer auf seinem Rechner ausgeführt werden.
Der Rank-Ausdruck erlaubt es dem Besitzer eines Rechners, zu definieren, welche Jobs sein Rechner bevorzugt ausführen soll. Er kann dabei sämtliche Attribute der Job ClassAd's verwenden. Gibt es nun mehrere Jobs, die auf dem Rechner ausgeführt werden könnten, wird der Rank-Ausdruck des Rechners für jeden der Jobs ausgewertet und der Job mit dem höchsten Rank dem Rechner zugewiesen. Darüber ist es dem Besitzer eines Rechners zum Beispiel möglich, dass er bevorzugt seine eigenen Jobs oder Jobs seiner Forschungsgruppe ausführen lässt. In der Standardkonfiguration ist der Rank für alle Jobs gleich.

6.2 Ablauf der Jobausführung

Der Status eines Rechners wird bei Condor in verschiedene Zustände und Aktivitäten in diesen Zuständen eingeteilt.
Am Anfang befindet sich jeder Rechner im Zustand Owner und hat die Aktivität idle. Dies heißt, er steht nicht für Jobs zur Verfügung und führt auch keinen Job aus. Wird nun der vom Besitzer definierte Start-Ausdruck lokal wahr, weil der Rechner zum Beispiel längere Zeit nicht genutzt wurde, geht er in den Zustand Unclaimed, was bedeutet, er steht für Condor zur Verfügung und wartet darauf, dass er einen Job zugewiesen bekommt. Die normale Aktivität in diesem Zustand ist idle, da der Rechner noch keinen Job zur Ausführung hat. Außerdem kann der Rechner in diesem Zustand noch kurzzeitig die Aktivität benchmarking ausführen, um die Floating Point und Intager Leistung des Rechners zu bestimmen. Sollte der Start-Ausdruck wieder FALSE werden, bevor der Rechner einen Job zugewiesen bekommt, geht er in den Zustand Owner zurück.
Hat der Rechner einen Job zugewiesen bekommen, geht er in den Zustand claimed und nimmt mit dem Start des Jobs die Aktivität busy auf. Sollte nun während der Ausführung der Start-Ausdruck FALSE werden und damit die Bedingungen für die Ausführung von Jobs auf dem Rechner nicht mehr gegeben sein, wird der Suspend-Ausdruck geprüft. Ist dieser TRUE wird die Jobausführung gepaust bis entweder der Start-Ausdruck wieder TRUE ergibt und somit mit der Ausführung des Jobs fortgefahren werden kann oder der Suspend-Ausdruck FALSE wird. Dies geschieht in der Standardkonfiguration, nachdem der Job 10 Minuten gepaust wurde. Ist der Suspend-Ausdruck FALSE geworden, geht der Rechner in den Zustand Preempting über, was bedeutet, dass der Job vom Rechner entfernt wird. Dort nimmt er zuerst die Aktivität vacating auf, in der versucht wird, einen Checkpoint zu schreiben. War das Schreiben des Checkpoints erfolgreich, geht der Rechner in den Zustand Owner und wird somit nicht mehr von Condor verwendet. Wird im Zustand preemting der Preempt-Ausdruck FALSE, was geschehen kann, weil von dem Job keine Checkpoints angelegt werden können oder weil das Anlegen des Checkpoints zu lange dauert, nimmt der Rechner die Aktivität Killing an wobei der Job ohne Speicherung eines Checkpoints vom Rechner entfernt wird und der Rechner wieder in den Zustand Owner geht.

6.3 User Prioritäten

Condor unterscheidet bei der Zuteilung von Jobs auf freie Rechner auch zwischen Prioritäten der User. Dabei gibt es zwei verschiedene Prioritäten. Die erste ist die Real User Priority (RUP). Diese spiegelt den Ressourcenverbrauch des Users wieder. Sie startet bei 0.5 und nähert sich allmählich der Anzahl der vom User belegten Ressourcen, also der Anzahl der Rechner, die Jobs des Users bearbeiten, an. Belegt der User wieder weniger Ressourcen, baut sie sich wieder ab. In der Standardkonfiguration geschieht dies mit einer Halbwertszeit von einem Tag.
Als zweites gibt es die Effectiv User Priority (EUP). Diese wird für die Resourcenvergabe verwendet. Je niedriger diese ist, um so mehr Ressourcen kann ein User zugeteilt bekommen. Sie ergibt sich aus der Real User Priority des Users durch die Multiplikation dieser mit seinem Prioritätsfaktor. Der Prioritätsfaktor kann für jeden User extra festgelegt werden. Sollte kein Prioritätsfaktor für den User definiert wurden sein, hat er den Faktor „Eins“.
Zusätzlich dazu wird die EUP mit einem hohen Faktor multipliziert, wenn der Job als „nice“ deklariert wurde, oder wenn er mittels flocking aus einem andern Pool in Auftrag gegeben wurde.

7. Sicherheit

7.1 Access Levels

Sämtliche Zugriffe auf die Condor Daemons werden in verschiedene Access Levels eingeteilt. Für diese Access Levels lassen sich alle Sicherheitseinstellungen einzeln treffen. Dies ermöglicht es, die administrativen Zugriffe wie das Starten und Beenden von Condor oder Änderungen der Konfiguration deutlich besser abzusichern als einfache Statusabfragen.
Das Administrator Level umfasst das Starten und Beenden von Condor auf den Rechnern des Pools und das Ändern der Userprioritäten. Das Owner Level ist für den Besitzer des Rechners gedacht. Dies beinhaltet zum Beispiel das Beenden von Jobs, die gerade auf dem Rechner ausgeführt werden. Veränderungen an der Konfiguration der laufenden Daemons fallen unter das Config Level. Das Negotiator Level enthält die Zuweisungen von Jobs auf die Rechner. Hier gibt es noch die Untergruppe Negotiator_schedd für den Zugriff auf Negotiatoren anderer Pools per flocking. Sämtliche Statusabfragen wie zum Beispiel die Maschine ClassAd's der Rechner des Pools oder die Job Queue fallen unter das Read Level. Jobs in Auftrag stellen fällt unter das Level Write, wobei dafür auch Read Access benötigt wird, der nicht automatisch mit gewährt wird. Bei Read gibt es noch die Untergruppen Read_Collector und Read_Startd, bei denen zusätzlich zu den Rechnern des Pools auch Zugriff auf Pools in die Jobs per flocking in Auftrag gegeben werden sollen, gewährt werden muss. Beim Write Level gibt es als Unterlevel noch Write_Collector und Write_Startd, bei denen zusätzlich zu den Rechnern des eigenen Pools den Rechnern Zugriff gewährt werden muss, die Jobs per flocking in den Pool stellen dürfen. Für die Kommunikation der Daemons untereinander, wie zum Beispiel das Anmelden am Collector oder Anfragen an den Collector, um die IP und den Port eines anderen Daemons herauszufinden, gibt es noch das Daemon Access Level.

7.2 Host / User based Access

Über die Host based Security kann festgelegt werden, von welchen Rechnern aus Zugriffe mit einem bestimmten Access Level erlaubt sind. In neueren Condor Versionen ist dies zu einem User basierten Filtern erweitert worden, wo festgelegt werden kann, welcher User von welchem Rechner aus zugreifen darf. Dies geschieht durch die Definition einer Allow und Deny Regel mit einer Liste der Rechner bzw. Rechner-User Kombinationen. Dabei sind auch Wildcards erlaubt, wobei bei Konflikten Einträge ohne Wildcards Vorrang haben.
In der Standardkonfiguration sind Administrator- und Negotiator-Zugriffe nur vom Central Manager erlaubt. Owner Zugriffe sind vom Central Manager und vom lokalen Rechner aus erlaubt. Config Zugriffe sind gar nicht erlaubt und Read und Write sind von allen Rechnern gestattet.

7.3 Server / Client Authentifizierung

Die Server/ Client Authentifizierung dient dazu, die Identität eines Users abzusichern, wenn dieser Anfragen an die Condor Daemons stellt, also zum Beispiel einen Job in Auftrag gibt. Dazu gibt es verschiedene Methoden. Auf Unix Systemen wird als Standard Methode FS verwendet. Bei FS wird der Client vom Server aufgefordert, eine bestimmte Datei auf der Festplatte anzulegen, und der Server überprüft dann den Eigentümer der Datei. Eine kleine Abwandlung davon ist FS-Remote, wo die Datei auf einem shared Filesystem angelegt werden muss. Bei NT Systemen wird als Standard Methode NTSPI verwendet, was die NT Lan Manger Funktionalität zur Authentifizierung verwendet. Des weiteren ist die Authentifizierung über Kerberos Zertifikate oder x.509 Zertifikate möglich. Als letztes gibt es noch die Möglichkeit, als Authentifizierungsmethode Claimtobe zu definieren, was heißt, dass der Server dem Client vertraut und keine Überprüfung der Identität stattfindet.
Es können mehrere Methoden zur Authentifizierung definiert werden, und Server und Client einigen sich dann auf eine, die sowohl beim Client als auch beim Server erlaubt ist. Dabei kann man definieren, dass bestimmte Methoden entweder nötig sind, bevorzugt werden, möglich sind oder nicht möglich sind. Dies gibt die Möglichkeit, zu Clients kompatibel zu bleiben, die bestimmte Authentifizierungsmethoden nicht können, aber diese zu verwenden, falls der Client sie kann.

7.4 Nachrichtenverschlüsselung

Condor kann Nachrichten, die über das Netzwerk verschickt werden, verschlüsseln. Dies geschieht entweder über Triple DES oder über Blowfish.
Ob eine Nachrichtenverschlüsselung stattfinden soll, kann für jedes Access Level einzeln definiert werden. Dabei ist es möglich, festzulegen, ob die Verschlüsselung für dieses Level unbedingt nötig ist, wenn möglich durchgeführt werden soll, nur stattfinden soll, wenn der Kommunikationspartner dies wünscht, oder nie verschlüsselt werden soll. Dies erlaubt es auch, ältere Condor Versionen im Pool zu haben, die noch keine Verschlüsselung durchführen können, aber wo es möglich ist Verschlüsselung zu verwenden. Genauso könnte man die Verschlüsselung nur auf Rechnern als notwendig definieren, die sich außerhalb des eigenen Netzwerkes befinden.

7.5 Nachrichtenintegrität

Um abzusichern, dass die Nachrichten zwischen den Rechnern des Pools nicht verfälscht werden, kann Condor eine Überprüfung der Integrität mittels md5 Checksummen durchführen. Ob dies durchgeführt werden soll, lässt sich genauso wie die Verschlüsselung für die unterschiedlichen Access Levels einzeln definieren. Auch hier kann wieder zwischen notwendig, bevorzugt, möglich und nicht möglich unterschieden werden.

8. Die Condor Universes

8.1 Standard Universe

Das Standard Universe unterstützt alle Features, die Condor bietet. Es können Checkpoints angelegt werden, und es sind Remote Sytem Calls möglich. Dafür gibt es im Standard Universe viele Anforderungen, die der Job erfüllen muss.
Zunächst müssen alle Jobs, die im Standard Universe laufen sollen, mit den Condor Libraries neu gelinkt werden. Die Jobs im Standard Universe dürfen keine Kernel Level Threads besitzen, also keine fork, exec, oder system Aufrufe verwenden. Des weiteren ist im Standard Universe keine Interprozesskommunikation wie zum Beispiel Pipes, Semaphoren oder shared Memory erlaubt. Die Programme dürfen auch keine Timer, Alarme oder Sleeps enthalten. Die Verwendung einer graphischen Oberfläche in den Programmen ist auch nicht gestattet. Files sollten von den Programmen nur read- oder write only geöffnet werden, da es sonst, wie schon bei den Checkpoints beschrieben, zu Problemen beim Fortsetzten kommen kann. Netzwerkkommunikation über Sockets ist den Programmen zwar gestattet, diese sollte jedoch begrenzt sein, da, solange noch Sockets geöffnet sind, keine Checkpoints angelegt werden.

8.2 Vanilla Universe

Das Vanilla Universe ist für ganz normale Programme gedacht, die nur als Binary vorliegen und nicht neu gelinkt werden können oder aber die Anforderungen des Standard Universe nicht erfüllen.
Im Vanilla Universe können keine Checkpoints geschrieben werden und es sind keine Remote System Calls möglich. Programme im Vanilla Universe sind also, wenn sie zusätzliche Dateien benötigen, auf ein gemeinsames Dateisystem angewiesen, oder der Auftraggeber muss im Submit File alle benötigten Files mit angeben, damit diese von Condor mit zum ausführenden Rechner übertragen werden. Dafür gibt es beim Vanilla Universe allerdings keine weiteren Einschränkungen für das Programm. Threads und Interprozesskommunikation sind hier erlaubt. Das Programm darf auch eine graphische Oberfläche verwenden. Ebenso können Shell Scripte als Jobs im Vanilla Universe ausgeführt werden.

8.3 Java Universe

Das Java Universe ist für die Ausführung von Java Programmen in einem Condor Cluster gedacht. Die Programme werden in der Java Virtual Maschine des Execute Rechners ausgeführt. Checkpointing und Remote System Calls sind dabei nicht möglich. Es ist aber möglich, Files anzugeben, die zum Ausführen übertragen werden sollen. Zusätzlich dazu können jar Files mit Klassen angegeben werden, die das Programm benötigt und die dann mit auf den ausführenden Rechner übertragen und dort geladen werden.

8.4 PVM Universe

Bei PVM handelt es sich um ein Softwaresystem zum Aufbau eine Parallelrechners aus normalen PC's. Im PVM Universe kann Condor Programme, die für eine PVM Umgebung entwickelt wurden, ausführen. Es ist dabei keine Änderung der Programme und kein neues Kompilieren oder Linken notwendig. Allerdings kann Condor nur Programme, die nach den Master-Worker Paradigma aufgebaut sind, ausführen. Dabei gibt es einen Master, der einen Überblick über sämtliche zu erledigenden Teilaufgaben hat und für diese Teilaufgaben Worker anfordert. Wenn diese Programme von Condor ausgeführt werden, wird der Master auf dem auftraggebenden Rechner gestartet. Dieser kann dort nicht gepaust oder beendet werden. Wenn der Master Worker anfordert, bekommt er freie Rechner des Condor-Pools als solche zugewiesen. Diese können allerdings jederzeit wieder wegfallen, sollten sie wieder von ihrem Benutzer verwendet werden.
Der PVM Support ist bei der Grundinstallation von Condor nicht dabei und muss zusätzlich installiert werden.

8.5 Globus Universe

Das Globus Universe dient dazu von einem Condor Cluster aus Jobs in Auftrag zu geben, die in einem Globus Grid laufen sollen. Condor kümmert sich dann darum, dass die Jobs an den Grid weitergeleitet werden und auch alle dazugehörigen Dateien zum Grid transferiert werden.

9. Erweiterung der Condorpools

9.1 Flocking

Durch Flocking soll es ermöglicht werden, mehrere Condor-Pools zusammenzuschließen. Es ist aber auch möglich, dass nur ein einzelner User eines Pools noch alternative Pools hat, in denen seine Jobs ausgeführt werden, wenn die Ressourcen im eigenen Pool nicht ausreichen.
Um das Flocking zu ermöglichen, müssen in der Konfiguration die Central Managers der Pools eingetragen werden, in denen Job durch Flocking ausgeführt werden sollen, und im Zielpool muss es den Rechnern erlaubt werden, dort Aufträge per Flocking auszuführen. Wird nun ein Job von einem User in Auftrag gegeben, fragt der lokale Schedd erst im eigenen Pool nach, ob Ressourcen für den Auftrag vorhanden sind. Ist dies nicht der Fall, geht er die Liste der Pools durch, in denen per Flocking Jobs ausgeführt werden sollen, bis einer Ressourcen für den Job frei hat. Im allgemeinen bekommen Jobs, die per Flocking aus einem andern Pool kommen, allerdings eine wesentlich niedrigere Priorität und könnten so auch während der Ausführung abgebrochen werden, falls User des eigenen Pools die Ressourcen benötigen.
Das Flocking stellt keine besonderen Anforderungen an die Jobs und geschieht automatisch, ohne dass der Auftraggeber eines Jobs dies für jeden Job extra definieren muss.

9.2 Condor_G

Bei Globus handelt es sich um ein Forschungsprojekt, das sich mit dem Aufbau von Grids, den dazugehörigen Infrastrukturen und den Problemen bei der Entwicklung von Anwendungen für Grids beschäftigt. Daraus hervorgegangen ist das Globus Toolkit, was Software zum Aufbau eines Grids und Tools zur Entwicklung von Anwendungen für diese Grids beinhaltet.
Mit Hilfe von Condor_G ist es möglich, solche Globus Grids als Ressourcen in den Condor Cluster einzubinden.
Dabei kommen die in Auftrag gegebenen Job erst wie normalen Jobs in die Job Queue von Condor. Um diese auszuführen, wird dann der Grid Manager gestartet, der den Job an den Globus Gatekeeper weitereicht und sich darum kümmert, dass alle für die Jobausführung notwendigen Dateien zum Grid transferiert werden.
Damit sind aber nur normale für den Grid ausgelegte Programme möglich. Es sind keine Jobs des Standard Universe möglich und somit auch kein Checkpointing. Um dies zu ermöglichen gibt es die GlideIn Jobs. Dabei werden die Condor Daemons im Globus Universe auf dem Grid gestartet. Dadurch wird der Grid zu einer normalen Ressource im Condor Pool und die Jobs können im Standard Universe in dem Grid ausgeführt werden.

Literatur

Condor Version 6.4.3 Manual
http://www.cs.wisc.edu/condor/manual/v6.4/

John Bent, Douglas Thain
„Condor Tutorial GGF-5 / HPDC-11"
http://www.cs.wisc.edu/condor/tutorials/condor-hpdc11.ppt

Peter Couvares, Todd Tannenbaum
„Condor Tutorial First EuroGlobus Workshop June 2001"
http://www.cs.wisc.edu/condor/slides/euroglobus-tutorial/euroglobus-tutorial.ppt

Scott Fields
„HUNTING FOR WASTED COMPUTING POWER"
http://www.cs.wisc.edu/condor/doc/WiscIdea.html

Derek Wright
„Condor Tutorial for Users INFN-Bologna"
http://www.cs.wisc.edu/condor/tutorials/infn-users/

Quellcode der Testprogramme

Kontinuierliches Schreiben:

```
#include <stdio.h>
#include <string>
#include <stdarg.h>
#include <math.h>


//Output Funktion, die wenn ein File Pointer da ist in ein File
//ansonsten nach stdout schreibt
void out(FILE * file,const char *fmt, ...)
{
	if(fmt == NULL)
		return;
	va_list ap;
	va_start(ap, fmt);

		if ( file == NULL )
			vprintf(fmt, ap);
		else
			vfprintf(file , fmt, ap);

	va_end(ap);
}

//Funktion, die nen weng sinnlos rumrechnet
void calculate(int cyclenumber,int speed)
{
	double value = 1;
	for ( int i=0;i < speed;i++ )
	{
		for ( int j=0;j<45000;j++)
			value = log( value/i * log( cyclenumber)  );
	}
}


int main(int argc, char* argv[])
{
	int cycles =  1;
	int speed = 100;
	std::string outputfile = "";
	FILE* output = NULL;

	//als Erstes die Argumente einlesen

	//erstes Argument sind die zu durchlaufenden zyklen
	if (argc > 1 )
		cycles = atoi( argv[1] );

	//Speedfactor, also wie stark müssen wir bremsen
	if (argc > 2 )
		speed = atoi( argv[2] );
```

```
	//Output File Name falls in File ausgegeben werden soll
	if (argc > 3 )
	{
		outputfile = argv[3];
		output = fopen(outputfile.c_str(),"w");
	}

	//so, nun geben wir erstmal die Argumente aus
	out( output , "Cycles:\t%i\n",cycles);
	out( output , "Speed:\t%i\n",speed);
	out( output , "File:\t%s\n",outputfile.c_str());

	out( output , "\nBegin Calculation\n\n");

	//Berechnungszyklus
	for (int i = 0; i<cycles; i++)
	{
		calculate(i,speed);
		out( output,"Cycle %i complete\n",i);
	}


	if ( output != NULL )
		fclose( output );
	return 0;
}
```

Ausgabefile appenden

```
#include <stdio.h>
#include <string>
#include <stdarg.h>
#include <math.h>

//Output Funktion, die wenn ein Dateiname da ist, ein File apanding
//oeffnet ansonsten nach stdout schreibt
void out(std::string file,const char *fmt, ...)
{
	FILE *output = NULL;

	if ( file.length() > 0 )
		output = fopen(file.c_str(),"a");

	if(fmt == NULL)
		return;
	va_list ap;
	va_start(ap, fmt);

		if ( output == NULL )
			vprintf(fmt, ap);
		else
			vfprintf(output , fmt, ap);
	va_end(ap);

	if ( output != NULL )
		fclose( output );
}
```

```
//Funktion, die nen weng sinnlos rumrechnet
void calculate(int cyclenumber,int speed)
{
      double value = 1;
      for ( int i=0;i < speed;i++ )
      {
            for ( int j=0;j<45000;j++)
                  value = log( value/i * log( cyclenumber)  );
      }
}


int main(int argc, char* argv[])
{
      int cycles =  1;
      int speed = 100;
      std::string outputfile = "";

      //als Erstes die Argumente einlesen

      //erstes Argument sind die zu durchlaufenden Zyklen
      if (argc > 1 )
            cycles = atoi( argv[1] );

      //Speedfactor, also wie stark müssen wir bremsen
      if (argc > 2 )
            speed = atoi( argv[2] );

      //Output File Name, falls in File ausgegeben werden soll
      if (argc > 3 )
      {
            outputfile = argv[3];
      }

      //so nun geben wir erstmal die Argumente aus
      out( outputfile , "Cycles:\t%i\n",cycles);
      out( outputfile , "Speed:\t%i\n",speed);
      out( outputfile , "File:\t%s\n",outputfile.c_str());

      out( outputfile , "\nBegin Calculation\n\n");

      //Berechnungscyclus
      for (int i = 0; i<cycles; i++)
      {
            calculate(i,speed);
            out( outputfile,"Cycle %i complete\n",i);
      }

      return 0;
}
```