Ressourcenverwaltung mit Load Sharing Facility

Daniela-Maria Pusinelli
pusinelli@cms.hu-berlin.de

Computeservice, LINUX-Cluster, Batch, Fair-Share Scheduler, LSF, N4000, ES40

Die zentrale Nutzung von Computer-Ressourcen erfordert die automatische Verwaltung und Vergabe der zur Verfügung stehenden Ressourcen. Dazu dienen Batch-systeme, die Rechenaufträge sammeln, nach den Anforderungen sortieren und auf die zur Verfügung stehenden Server verteilen. LSF ist so ein Ressourcenmanagement, das in einem heterogenen Cluster von Servern mit unterschiedlicher Anzahl von Prozessoren arbeitet. Es verfügt über einen Fair-Share Scheduler.

Charakteristik von LSF

Bereits seit vielen Jahren wird die konkurrierende Nutzung von Computer-Ressourcen im CMS über Software-Werkzeuge (Batchsoftware) gesteuert, die es erlauben, allen beteiligten Nutzern gleichgewichtig und nach Bedarf angeforderte Ressourcen zuzuteilen. Diese Batchsoftware hat sich im Laufe der Jahre der Entwicklung der Computerhardware angepasst. Während Anfang der 90er-Jahre NQS auf den Rechnern FX/2800 (Alliant) und C3820 (Convex) noch mehr eine lokale Funktion hatte, so ist die heutige Batchsoftware für das Grid-Computing gerüstet. Sie verwaltet ein ganzes Netz von Computern und deren Ressourcen. Eine Reihe von ihnen ist Open Source. So ist z. B. Sun Grid Engine (SGE) [1], der Nachfolger der von Genias entwickelten Software Codine, heute auf den Plattformen SUN und LINUX frei verfügbar. Codine war mehrere Jahre als kommerzielles Produkt im CMS auf unterschiedlichen Plattformen im Einsatz. Ebenso ist OpenPBS [2] ein freies Softwareprodukt für LINUX, das zusammen mit dem frei verfügbaren Scheduler MAUI [3] sehr häufig auf Beowulf-Clustern zum Einsatz kommt.

Im CMS fiel die Entscheidung für eine einheitliche Ressourcenverwaltung auf das kommerzielle Produkt LSF [4] von der Firma Platform Computing Corp. in Kanada. In Deutschland wird die Software durch die Firma Science + Computing AG in Tübingen bereitgestellt. LSF war bereits seit längerer Zeit auf dem HP-Server N4000 als lokale Installation im Einsatz. Es besticht durch seine hohe Funktionalität und seine flexible Nutzung, besonders in einem heterogenen Cluster aus Rechnern verschiedenster Hersteller und Leistungsfähigkeit und mit einer stark variierenden Anzahl von Prozessoren. Ein gleichzeitiges Rechnen von seriellen und parallelen Jobs innerhalb eines LSF-Clusters ist gut möglich.

LSF-Cluster im CMS

Mit der Installation des LINUX-Cluster wurde ein neues LSF-Cluster CLOU (**CL**uster **O**f **U**nix machines) aufgebaut, in dem die bereits vorhandenen Computeserver, der HP-Server N4000 (octus), der COMPAQ-Server ES40 (quadro), sowie das neue LINUX-Cluster (clan) als eine Compute-Ressource zusammengefasst werden. Ein Nutzer, der einen Auftrag rechnen möchte, kann sich auf einem beliebigen Server des Clusters einloggen und von dort den Auftrag mit den Ressourcenanforderungen an das LSF-Cluster CLOU mittels eines Job-Skriptes abschicken. Ein Einloggen auf dem gewünschten Rechner ist nicht mehr notwendig. Der LSF-Master versucht dann, die geeignete Compute-Ressource im LSF-Cluster ausfindig zu machen und den Auftrag an eine eventuell freie Ressource zu übergeben. Falls das nicht möglich ist, wird der Auftrag in eine Warteschlange (Queue) eingereiht, bis die entsprechende Ressource frei ist. Der LSF-Master sowie der Scheduler des LSF-Clusters CLOU laufen auf dem Master des LINUX-Clusters. Das ist notwendig, da die Knoten des LINUX-Clusters außerhalb nicht bekannt sind, sondern ihnen nur dynamische Hostadressen (DHCP) zugeordnet sind. Auf allen Computeservern des LSF-Clusters laufen der Remote Execution Dämon, der Load Information Dämon sowie der Slave Batch Dämon. Alle diese werden beim Start der Server aktiviert und sind während der gesamten Systemlaufzeit der Server aktiv.

Mit dem Kommando `lsload` erhält der Nutzer einen genauen Überblick über die Auslastung der einzelnen Server.

Queuekonfiguration

Das LSF-Cluster CLOU bietet durch die unterschiedliche Ausstattung der Rechner eine breite Palette von Anwendungsmög-

Queue	Server	Betriebssystem	Prozessoren	Runlimit	Memory limit	Priorität
compile	clan	LINUX	1–2	30 m	1 GB	90
par	clan	LINUX	4–32	24 h	3 GB	70
normal	clan	LINUX	1–2	12 h	2 GB	85
quadro_mpi	quadro	Tru64 UNIX	2–4	24 h	3,8 GB	50
quadro	quadro	Tru64 UNIX	1	24 h	1,5 GB	40
octus_mpi	octus	HP-UX 11.0	4–8	24 h	12 GB	65
octus_big	octus	HP-UX 11.0	1	24 h	2 GB	55

Tab. 1: Queuekonfiguration des LSF-Clusters CLOU

lichkeiten. Es können sowohl parallele als auch serielle Berechnungen ausgeführt werden.

Für Programme, die auf SMP-Rechnern entwickelt wurden, sind die Queues `octus_mpi` und `quadro_mpi` geeignet. Für Programme, die mit MPI parallelisiert wurden und die gut skalieren, ist die Queue `par` zu wählen. Die seriellen Programme sind an die Queues `normal`, `octus_big` oder `quadro` zu schicken. Das Kommando `bqueues` listet alle Queues in einer Kurzform auf. Es zeigt auch die Prioritäten der einzelnen Queues sowie ihren Status (open, closed, activ oder inactiv). Eine exakte Auflistung aller Parameter z. B. der Queue `par` liefert das Kommando `bqueues -l par`.

Eine inaktive Queue lässt bereits rechnende Jobs weiterlaufen, bis sie beendet sind, aber kein neuer Job wird gestartet. Die Jobs verbleiben in der Warteliste, bis die Queue wieder aktiv wird. Eine geschlossene Queue nimmt überhaupt keine Jobs an. Nutzer einer Arbeitsgruppe werden auch unter LSF zu einer Gruppe zusammengefasst. Das Kommando `bugroup` listet die Nutzer aller Gruppen. Ein zugelassenes Kennzeichen muss in einer dieser Gruppen eingetragen sein. Anderenfalls kann unter dem Kennzeichen kein Job an das LSF-Cluster abgeschickt werden. Die Queues `compile` und `normal` sind für alle zugelassenen Nutzer des LSF-Clusters offen. Ist beispielsweise die Arbeitsgruppe `filippou` nicht in der Queue `par` eingetragen, kann kein Mitarbeiter der Arbeitsgruppe die Queue benutzen. Der Vorrang gewisser Queues wird über die Priorität der Queue gesteuert. Die Queue `octus_mpi` ist »preemptive« zur Queue `octus_big`, d. h. Jobs der erstgenannten Queue werden bevorzugt abgefertigt. Außerdem wirkt der Fair-Share Scheduler, d. h. die Jobs beider Queues laufen über eine gemeinsame Share-Verwaltung. Analog verhält es sich bei den Queues `quadro_mpi` und `quadro`. Auch die Queues `normal`, `compile` und `par` laufen über eine Share-Verwaltung. Alle Arbeitsgruppen haben die gleichen Shares. Eine Information über die verbrauchten Shares in den Queues kann sich jeder Nutzer mit dem Kommando `bqueues -l par` anzeigen lassen.

Eine Zusammenfassung der Queue-Parameter liefert die Tabelle 1.

Abschicken von Jobs

Es gibt drei Möglichkeiten, Jobs abzuschicken:

1. *Interaktiv*: Jobs werden interaktiv gestartet. Dazu wird das Kommando `bsub` aufgerufen und nach dem Prompt `bsub>` können entsprechende UNIX Kommandos oder LSF-Kommandos (`#BSUB` …) eingegeben werden.
2. *Skriptfile*: Das ist der Normalfall. Rechenaufträge werden in ein Skript gepackt und mit dem Kommando `bsub < script` abgeschickt.
3. *xbsub*: In diesem Fall wird das Abschicken des Jobs durch Verwendung des Grafischen User Interface erleichtert.

Die Queue, in die der Job gestellt werden soll, kann in der Kommandozeile oder im Skript angegeben werden. Die Angabe in der Kommandozeile überschreibt die des Skriptes. Alle Jobs, die nicht sofort abgearbeitet werden, kommen in die Liste der wartenden Jobs (Pending-Liste). Ein Job muss auf die Abarbeitung warten, wenn entweder der Grenzwert für die Anzahl der Jobs in der Queue oder wenn der Grenzwert für die Anzahl der Jobs, die der Nutzer im LSF-Cluster starten darf, bereits erreicht sind.

Der Nutzer erhält als Reaktion auf den Submit-Befehl die Antwort, dass der Auftrag akzeptiert wurde sowie die Nummer des Auftrages. Im Falle, dass die Parameter im Submit-Befehl nicht stimmen, z. B. die CPU-Zeit ist größer als die in der Queue erlaubte, die Anzahl der angeforderten Prozessoren ist geringer als in der Queue gefordert, kann die Ressourcenanforderung nicht erfüllt werden und der Auftrag wird nicht angenommen.

Schreibt der Nutzer in sein Skript, dass er zu Beginn und Ende des Auftrages informiert werden möchte, kann er sich anderen Aufgaben zuwenden, er bekommt eine E-Mail sobald der Job gestartet und eine, sobald er beendet ist.

LSF macht es möglich, dass eine variable Anzahl von Prozessoren angefordert werden kann, so dass der Job auch zur Abarbeitung kommt, falls weniger Prozessoren zur Verfügung stehen. Mit der Zeile `#BSUB -n 4,32` im Skript werden z. B. 4–32 Prozessoren angefordert. Das ist vorteilhaft, da unter Umständen Jobs, die 16 und mehr Prozessoren anfordern, sehr lange warten müssen, falls viele Jobs mit weniger Prozessoren in der Pending-Liste stehen.

Der Nutzer findet auf allen Rechnern des LSF-Clusters dieselbe Umgebung vor, d. h. sein Home-Verzeichnis ist auf allen Rechnern identisch. Da die Home-Verzeichnisse nur über einen geringen Speicherplatz verfügen, stehen auf allen Rechnern lokale `/scratch` Bereiche mit unterschiedlichen Kapazitäten zur Verfügung, die temporäre Daten aufnehmen können. Die Daten in den `/scratch` Bereichen unterliegen einem Kontrollmechanismus, sie sind dort nicht sicher gespeichert und müssen zur permanenten Ablage an anderer Stelle abgelegt werden.

Kontrolle der laufen Jobs

Die Kontrolle der laufenden Jobs erfolgt mit dem Kommando `bjobs`, es liefert Informationen zu:

`bjobs`	den eigenen Jobs
`bjobs -u all`	allen laufenden Jobs
`bjobs -u all -pl`	allen wartenden Jobs und den Grund des Wartens

Der Modifikation eines wartenden Jobs dienen die Kommandos:

`btop`	setzt den Job an die Spitze der wartenden
`bswitch`	lenkt den wartenden Job um in eine andere Queue
`bmod`	modifiziert einen wartenden Job

Für die Kontrolle der laufenden Jobs existiert auch ein GUI, die LSF Batch-Console `xlsbatch`. Die Abbildung 1 illustriert die Ausgabe des Kommandos. Das Abschicken von Jobs an das LSF-Cluster ist ausführlich in [5] beschrieben.

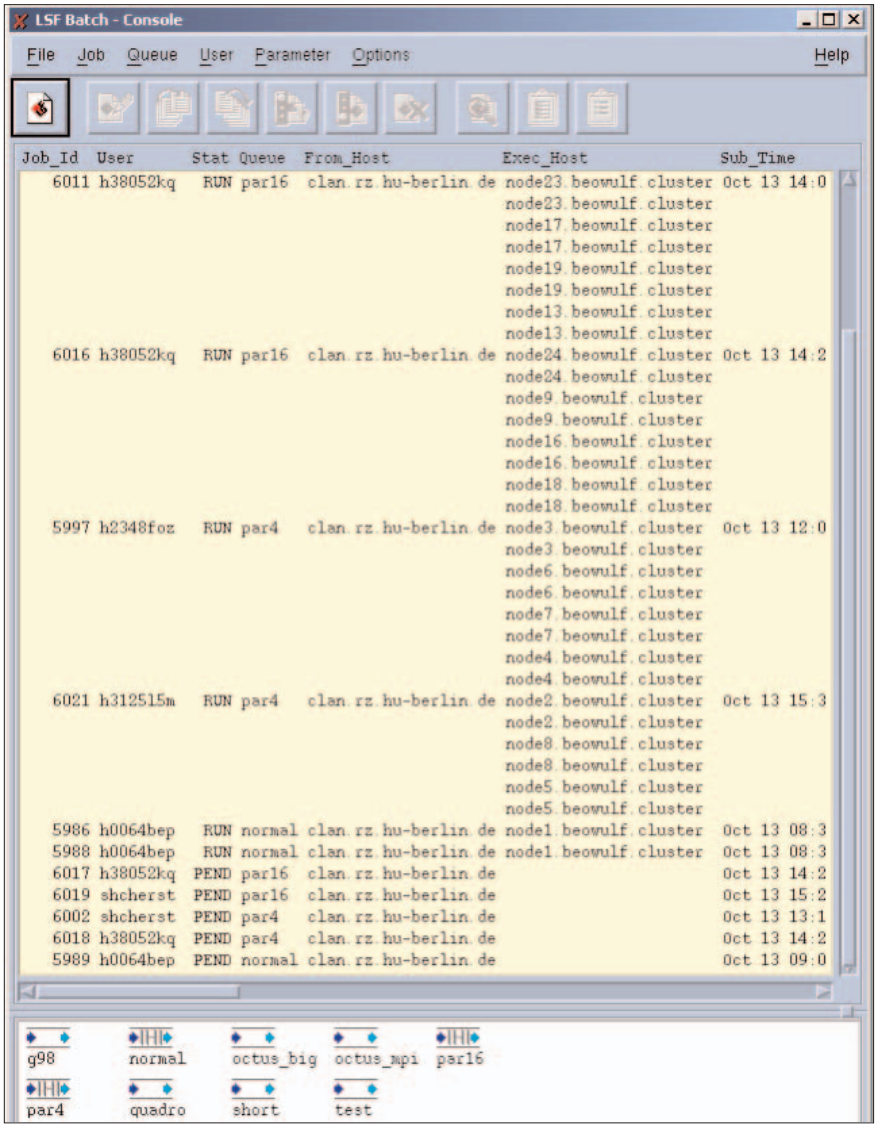

Abb. 1: Ausgabe des GUI xlsbatch

Beispiel

Ein Mitarbeiter aus dem Institut für Chemie möchte eine Trubomole Rechnung starten. Die Software ist auf den Servern `clan`, `quadro` und `octus` installiert, er kann sich also für einen der Server entscheiden, zumal die Turbomole Routinen parallel oder seriell gerechnet werden können. Die Wahl könnte sich eventuell an den freien Ressourcen orientieren. Er stellt fest, der Server quadro ist belegt, aber auf dem LINUX-Cluster `clan` hat die Queue für parallele Jobs noch freie Slots. Er schreibt neben seinen Anforderungen zur Laufzeit (12 h) und zum Speicher (1 GB) die Queue `par` in sein Skript und die Anzahl der Prozessoren (4–12), die er verwenden möchte. Eingeloggt ist der Nutzer gerade auf `quadro` und hat sein Skript als `run_turbo_test` im Home-Verzeichnis abgespeichert. Er hat auch eingetragen, dass er zu Beginn (B) und Ende (N) des Jobs informiert werden möchte. Mit `bsub < run_turbo_test` schickt er das Skript an das LSF-Cluster CLOU.

Nach Erhalt der E-Mail über das Ende des Jobs muss er sich dann erstmals auf dem Server `clan` einloggen und nach den Ergebnisdaten im Verzeichnis `/scratch/$USER` schauen.

```
#!/bin/bash
#BSUB -n 4,12
#BSUB -B
#BSUB -N
#BSUB -q par
#BSUB -J turbo_4
#BSUB -M 1000000
#BSUB -c 12
. /usr/local/bin/use-mpich124-intel.sh
cd /scratch2/${ USER}
cp /dev/null nodefile
set $LSB_HOSTS
for i
do
        echo ${ i} | cut -f 1 -d ».« >> nodefile
done
nnodes=$#
echo $nnodes; cat nodefile
export PARNODES=${ nnodes}
export P4_RSHCOMMAND=rsh
export HOSTS_FILE=/scratch2/${ USER} /nodefile
time mpirun -np $nnodes -machinefile $nodefile rdgrad_mpi
```

Abb. 2: Beispielskript Turbomole Rechnung

Literatur

[1] http://wwws.sun.com/software/gridware/
[2] http://www.openpbs.org/main.html
[3] http://www.supercluster.org/maui/
[4] http://www.platform.com/products/LSF/
[5] http://www.hu-berlin.de/rz/compsv/using_5.1/index.html