Batch-Service am RZ

Schon viel ist über die verschiedenen "Services" am Rechenzentrum geschrieben worden und nun auch noch *Batch-Service*!

Wozu ein Batch-Service?

Die großen Mainframes bieten den Nutzern eine immense Compute-Leistung, die natürlich genutzt sein will. Aber auch auf dem Gebiet der "Minis" wird inzwischen eine Menge an CPU-Leistung geboten. Meist werden Workstations noch zu sogenannten "Clustern" oder "Farmen" zusammengefaßt und verwaltet. Es geht also darum, die CPUs in einem Netzwerk nicht "rumidlen" zu lassen, sondern ihnen sinnvolle Arbeit zu geben, sowie eine ausgewogene Lastverteilung (Loadbalancing) auf den einzelnen Maschinen zu erreichen. Es muß also ein geeignetes Hilfsmittel her!
Ein Batchsystem (batch = Stoß) schickt stoßweise Requests (Nutzeraufträge) zur Bearbeitung los, die vorher von den Benutzern in entsprechende Warteschlangen (Queues) gestellt worden sind. Der Nutzer hat den Vorteil, nicht anwesend sein zu müssen, wenn seine Arbeit erledigt wird (wie herrlich!). Es wurden bisher viele Batchsysteme entwickelt, die abhängig von der Portabilität mehr oder weniger Verbreitung gefunden haben.

Was leistet NQS:

Als Industrie-Standard hat sich das NQS (Network Queueing System) herauskristallisiert, das ursprünglich von Sterling Software Incorporated im Auftrage der NASA entwickelt und, wie bei anderen öffentlich finanzierten Projekten auch, anschließend als Public Domain Produkt zur Verfügung gestellt wurde. Das NQS bietet die Möglichkeit, zwei Typen von Queues einzurichten:

1. Batch-Queues: Requests werden auf dem lokalen oder einem Rechner im Netzwerk bearbeitet.
2. Pipe-Queues: Requests werden zu einer Batch-Queue eines Rechners im Netz oder zu einer weiteren Pipe-Queue geschickt.

Den Batch-Queues können Limits, Prioritäten und andere Parameter zugeordnet werden, wie z.B.:

- Accounting on/off
- Access (un)restricted
- Checkpoint yes/no
- Run_limit: max. Anzahl gleichzeitig rechnender Requests einer Queue
- CPU_time_limit: max. zulässige CPU-Zeit für einen Job
- Nice_limit: min. nice Wert für einen Job
- Working_set_limit: max. physischer Speicher eines Jobs

Das NQS erlaubt es, Requests zu "checkpointen". Bei einem Shutdown des NQS werden alle Requests, die ein "checkpointen" erlauben, "gecheckpointet", so daß sie bei einem Restart des NQS ebenfalls wieder gestartet werden können, und zwar mit dem Bearbeitungsstand zum Zeitpunkt des Checkpoints.
Weiterhin ist es möglich, die Requests zu überwachen:

- Ändern der Priorität eines Requests
- Transportieren eines "non-running" Requests von einer Queue in eine andere
- Suspendieren eines "running" Requests
- Fortsetzen eines suspendierten Requests

Erste Erfahrungen mit dem NQS haben wir auf unserem Parallelrechner ALLIANT FX/2800 gemacht. Hier wurde das NQS aber nur als lokales Batchsystem genutzt, da bisher kein NQS auf den anderen Plattformen zur Verfügung stand.

ConvexNQS+ auf der Metaserie:

Auf der Metaserie (Convex C3820 und HP 9000/735) wird ein auf dem Standard NQS beruhendes Batchsystem ConvexNQS+ zur Verfügung gestellt. Damit soll das Zusammenspiel zwischen den Hosts der Metaserie ermöglicht werden. Als Hilfsmittel dient die Umgebungsvariable CLUSTER, die die entsprechenden Kommandos auf dem ausgewählten Cluster ausführt (siehe RZ-Mitteilungen 5/93).

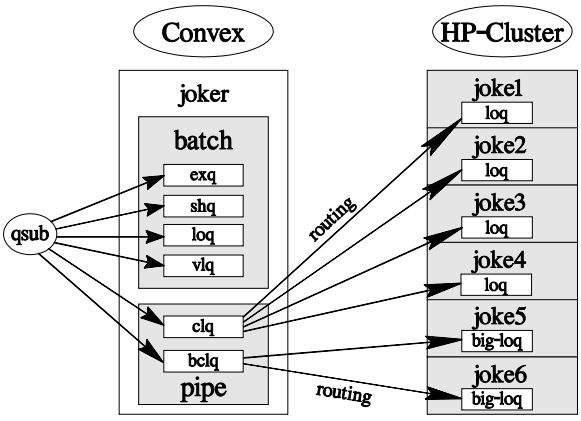

Abbildung 1

Die Abbildung 1 zeigt die derzeitige NQS-Konfiguration auf der Metaserie. Alle Requests, auch die für den HP-Cluster, werden in Queues der Convex gestellt.

Die Requests der 4 Batch-Queues:

	CPU	Memory
• **exq** = Express-Queue:	10 min	30 MB
• **shq** = Short-Queue:	30 min	50 MB
• **loq** = Long-Queue:	5 h	60 MB
• **vlq =** Very-long-Queue:	48 h	70 MB

rechnen auf der Convex.

Die Requests der Pipe-Queue **clq** (Cluster-Queue) werden zu den Clusterknoten **joke1** bis **joke4** geroutet. Die Requests der **bclq** (Big-Cluster-Queue, für Requests, die viel Hauptspeicher benötigen) werden zu den Clusterknoten **joke5** und **joke6** geroutet. Bei beiden Queues dient als Queue-Server /usr/lib/nqs/pipedemand, d.h. die Clusterknoten fordern sich die Requests von der Convex ab.
Die wichtigsten ConvexNQS+ Nutzerkommandos sind:
qsub, qdel, qstat, qps, qrestart, qlimit, qjlist, und **qchkpnt.**
Sie können alle in den entsprechenden Manuals nachgelesen werden. In allen **qsub** Kommandos können Parameter direkt übergeben werden, oder sie werden in entsprechenden Umgebungsvariablen definiert. Wichtig ist die Umgebungsvariable ENVIRONMENT, die beim Abschicken eines Requests auf "Batch" gesetzt ist. In den Startprozeduren, .profile für die Bourne-Shell bzw. .cshrc für die C-Shell, ist die Umgebungsvariable ENVIRONMENT abzutesten, so daß das Setzen von Terminaleigenschaften verhindert werden kann.
Eine Schwachstelle des ConvexNQS+ besteht darin, daß für die HP-Knoten die Vergabe von Limits eingeschränkt ist, d.h. kein CPU_time_limit und Working_set_limit sind setzbar. Aber eine erste Stufe eines netzweiten Batchsystems im Rechenzentrum ist damit gegeben. Ein weiterer Schritt soll dahin gehen, den HP-Pool sowie andere leistungsfähige Workstations zu integrieren. Dazu muß ein neues Hilfsmittel her. Das von der Softwarefirma GENIAS entwickelte System CODINE erscheint uns als geeignet.

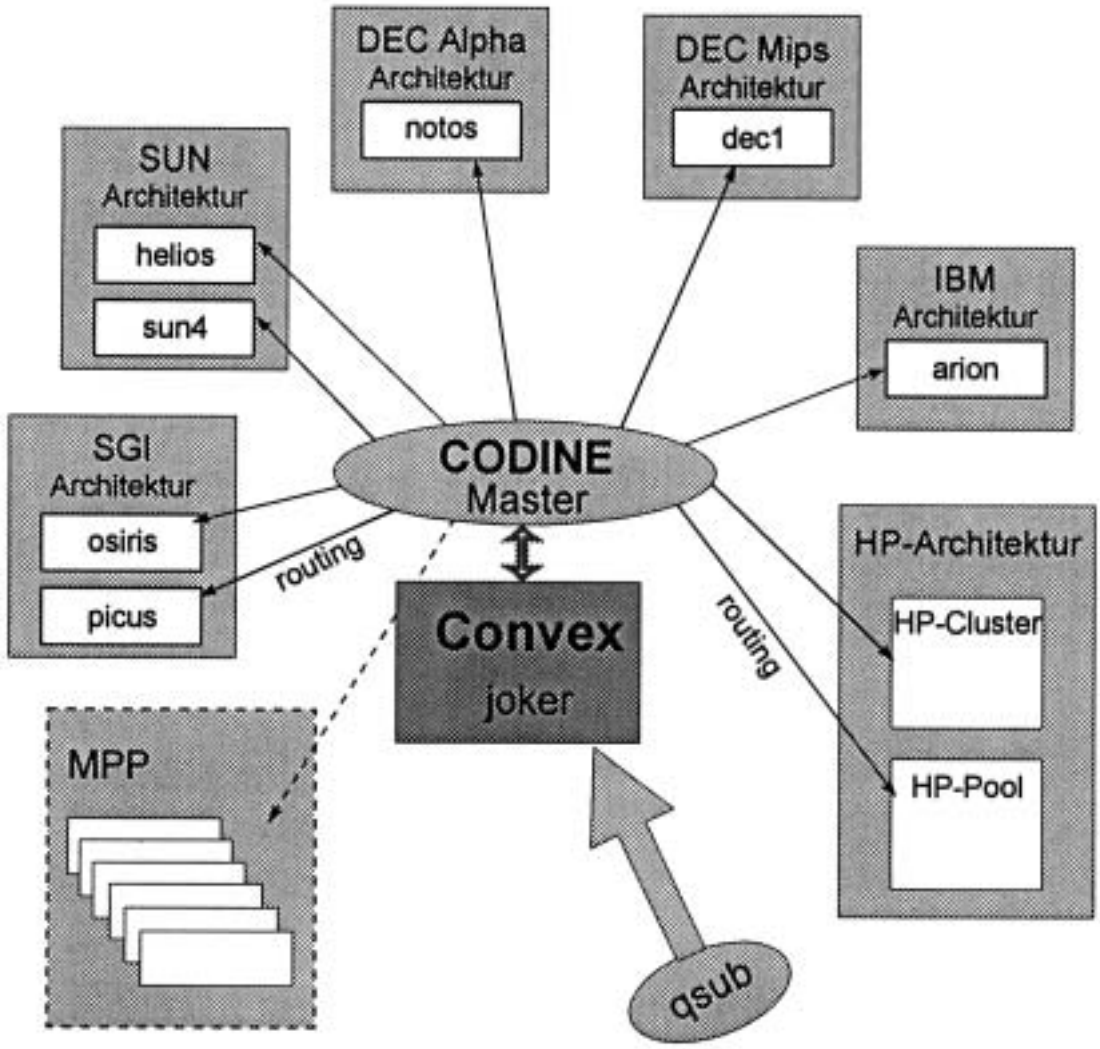

Abbildung 2: Mögliche CODINE-Konfiguration am RZ

Ausblick: CODINE

CODINE (COmputing in DIstributed Network Environment) beruht auf einer Client/Server-Struktur. Als Server dient ein Masterrechner (Informations-Server), der über den aktuellen Zustand und die Rechenlast jedes einzelnen CPU-Servers informiert. Die CPU-Server arbeiten unabhängig vom Master-Server, aber konsultieren ihn nach getaner Arbeit. Es ist auch möglich, den Masterprozeß von einer Maschine auf eine andere zu migrieren. Die Vorteile von Client/Server-Modellen liegen im Preis und in der Erweiterbarkeit. CODINE berücksichtigt die Bedingungen eines heterogenen Netzwerkes und versucht, die unterschiedliche Hardware optimal einzusetzen.

Was leistet CODINE?

Folgende Eigenschaften werden von CODINE unterstützt:

- Batch-, interaktive und parallele Jobs
- nutzergesteuertes und transparentes (Programme ohne exec, fork und sockets) Checkpointing
- ausgewogene Lastverteilung
- Ausführung von parallelen Jobs unter PVM, EXPRESS, p4 und Linda
- NQS Interface
- Accounting und Statistik
- DCE Technologie

Die CONDINE-Queues werden architekturbezogen und nach Gruppen sortiert. Die Queues können hierarchisch organisiert werden. Requests können zu einer speziellen Architektur, zu einer Gruppe oder zu einer speziellen Queue geschickt werden. CODINE wählt in den ersten beiden Fällen die geeignetste Queue im Netzwerk aus.
Als neue Kommandos kommen **qmod** und **qidle** hinzu, die eine Queue auf einer Plattform suspendieren, um diese dem Besitzer wieder für seine interaktive Arbeit zur Verfügung zu stellen. Die Requests der Queue werden in diesem Falle migriert. Das Kommando **qsh** eröffnet auf einem Host mit "leichter" Last eine interaktive Shell. **qconf** stellt das Nutzerinterface zum Arbeiten mit den Queues dar.
Eine mögliche Konfiguration von CODINE am Rechenzentrum der Humboldt-Universität könnte so aussehen, wie in Abbildung 2 dargestellt. Für parallele Jobs wird ein geeigneter Sub-Cluster von CODINE ausgewählt, nachdem der Nutzer die geforderte Anzahl von Queues über einen allgemeinen Request mitgeteilt hat. Hier ist der Begriff Queue mit Knoten gleichzusetzen.
CODINE bietet 3 verschiedene Nutzerinterfaces. Für die tägliche Routinearbeit ist das Einbetten der Kommandos in Job-Scripts sinnvoll. Das Kommandozeilen-Interface fügt sich den POSIX Anforderungen. Ein OSF/Motif Interface bietet die Möglichkeit, interaktiv auf alle Kommandos zuzugreifen. Ein graphisches Display, das Auskunft über die Nutzung der Cluster gibt, ist verfügbar.
Die HPCN-Kommission (High Performance Computing and Networking) der EG wird in den kommenden 2 Jahren die Weiterentwicklung von CODINE fördern, wobei diese in Richtung Metacomputing gehen soll. Die Perspektive für CODINE ist gegeben!

Daniela-Maria Subklew