Condor Benutzerdokumentation

Thomas Reusch, Hauke Hoffmann, Nico Maas

25. Februar 2016

Inhaltsverzeichnis

1 Einleitung

1.1 Hinweise

Dieses kleine HowTo dient als Bedienhilfe für das Queuing-System Condor. Wir beziehen uns hierbei ausschließlich auf die von uns vorkonfigurierte Condorinstallation. Grundkenntnisse zur Benutzung von Linux ohne grafische Oberfläche werden vorausgesetzt.
Sollte an dem Grundsystem (Ubuntu LTS) etwas geändert werden, ist zu beachten, dass sich Veränderungen bei der Kernelversion, glibc und GCC auf Condor auswirken können. Das offizielle Manual ist hilfreich, um detaillierte Informationen einzuholen, da dieses Dokument nur das nötigste Grundwissen zum Bedienen vermittelt. Die Konfigurationsdateien liegen auf den Maschinen vor und sind mit Kommentaren zum besseren Verständnis versehen. Ein Einblick in die Dateien und ein Editieren ist wie folgt möglich. Zu beachten ist, dass ein Bearbeiten der Dateien nur mit root-Rechten gestattet ist.

```
# vim /etc/condor/condor_config
```

bzw.:

```
# vim /etc/condor/condor_config.local
```

Sollten Fehler auftreten, sei es, dass Condor nicht (richtig) gestartet wird, oder dass Jobs nicht bearbeitet werden, wird die Fehlersuche durch Lesen der Logdateien vereinfacht. Die Logdateien, die von Condor für die Daemons angelegt werden, befinden sich in folgendem Pfad:

```
$ cd /var/condor/log/
```

Jobspezifische Logs für laufende (oder zumindest in die Queue übergebene) Jobs befinden sich hier:

```
$ cd /var/condor/execute/
```

1.2 Zugriff auf die Maschinen

1.2.1 Zugriff via Linux

Unter Linux funktioniert der Zugriff auf den Cluster, indem per SSH auf den Condorserver verbunden wird. Ein Zugriff auf die Clients ist weder vonnöten noch erwüscht, da das Verteilen der Jobs von Condor automatisiert erledigt wird. Eine Verbindung auf den Cluster funktioniert wie folgt, wobei „Benutzername“ durch den persönlichen Benutzernamen zu ersetzen ist.

```
$ ssh Benutzername@hpccluster.uni-trier.de
```

Sofern Dateien auf den Cluster zur weiteren Verarbeitung hochgeĺaden werden müssen, geschieht dies über „scp“. Aus Sicherheitsgründen gibt es keine (Samba-)Freigaben. Die hochzuladenen Dateien werden im home-Verzeichnis des Benutzers abgelegt; ein Befehl um Dateien hochzuladen könnte folgendermaßen aussehen:

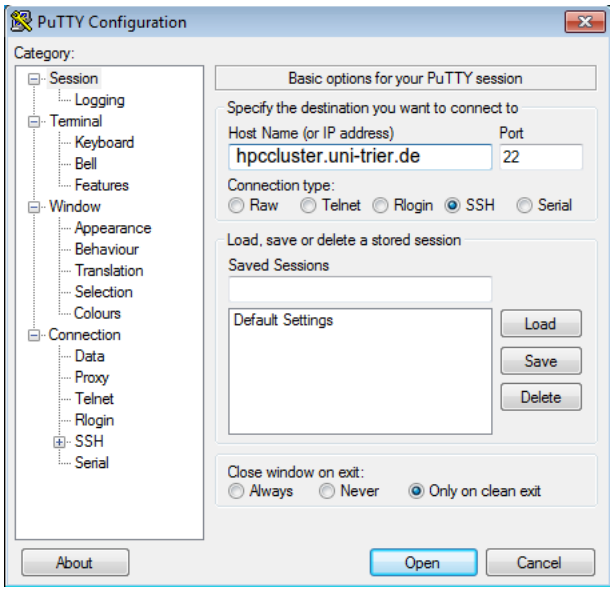
```
$ scp [Datei_zum_Hochladen] Benutzername@hpccluster.uni-trier.de:~/
```

1.2.2 Zugriff via Windows

Um von Windows aus auf den Cluster zuzugreifen, wird das Tool „PuTTY“ empfohlen. Falls nicht vorhanden, kann PuTTY kostenlos unter folgender Adresse heruntergeladen werden:

`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`

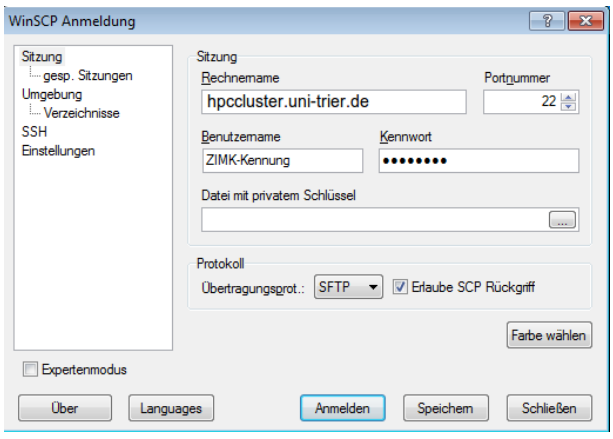
In das Feld „Host Name“ muss der Name „hpccluster.uni-trier.de“ eingetragen werden; die restlichen Einstellung können belassen werden (s. Screenshot). Nach Klick auf Öpen"wird Ihnen ein Login-Fenster angezeigt. Dort geben Sie die Ihnen mitgeteilte Kennung sowie Ihr Passwort (ZIMK-Passwort für Angehörige der Universität Trier) ein.



Unter Windows ist eine Dateiübertragung zum HPCcluster z.B. mit dem Programm WinSCP möglich. Falls nicht vorhanden, kann WinSCP kostenlos unter folgender Adresse heruntergeladen werden:

`http://winscp.net/eng/download.php`

Die Anmeldung erfolgt mit den selben Benutzerdaten wie bei PuTTY.



2 Arbeitsumgebung

Als Arbeitsumgebung des Condor-Clusters kommt lediglich ein Linux-System ohne grafische Oberfläche zum Einsatz. Die Vertrautheit mit diesem System wird vorausgesetzt.

Grundbefehle lassen sich in vielen Büchern mit Titeln wie Bash-Crashkurs oder Linux-Crashkurs aneignen. Eine kurze Einführung kann erfragt werden. In den folgenden Abschnitten wird daher nur die Verwendung von Condor beschrieben.

3 Submit-Dateien

3.1 Inhalt

```
Universe = vanilla
```

Hiermit ist das Condor-Universum gemeint. Vanilla ist hier immer eine gute Wahl, da im Vanilla-Universu alle Arten von Jobs laufen können (z.B. Shell-Scripte und proprietäre Software). Der Vollständigkeit halber weitere Universes:

- Standard
- MPI (Message Passing Interface)
- Java

```
Executable = /usr/bin/R
```

Executable gibt die auszuführende Datei an. Hier können beliebige Binaries oder Scripte angegeben werden.

```
Arguments = --vanilla
```

Hiermit gibt man Argumente an, die der Executable übergeben werden.

```
Transfer_executable = false
```

Hiermit gibt man an, ob die Datei, die bei EXECUTABLE angegeben ist, vom SUBMITTER auf die Rechenknoten übertragen werden soll. Default ist hier TRUE. Da auf unserem Cluster die R-Binary nur auf den Rechenknoten installiert ist, sollte hier bei Benutzung von R immer FALSE übergeben werden.

```
Requirements = Memory > 128
```

Mit dieser Zeile gibt man die „Bedürfnisse“ des Jobs an. Folgende Argumente sind semikolongetrennt außerdem möglich:

- Arch (==SUN4u)
- OpSys (==Solaris28)

```
transfer_input_files = foo.R, foo2.R
```

Je nachdem, wie der Pool nachher gestaltet wird, sollte hier entweder nichts angegeben werden (in dem Fall, dass es bei den globalen Homes bleibt), oder die Dateien, die zu Beginn des Jobs übertragen werden sollen.

```
stream_output = true
WhenToTransferOutput = ON_EXIT_OR_EVICT
```

Diese Parameter bewirken, dass die Ausgabe (stdout) des Jobs an den SUBMITTER zurückgestreamt wird.

Folgende Angaben beinhalten Pfade zu:

- der Error-Datei, die nach der Ausführung des Jobs den Inhalt von stderr enthält:

```
Error = foo.err
```

- der Log-Datei, die nach der Ausführung einen condor-generierten Job-Log enthält:

```
Log = foo.log
```

- der Input-Datei, die bei einem R-Job angegeben wird („R-Datei“):

```
Input = foo.R
```

- der Output-Datei, die nach der Ausführung den Inhalt von stdout enthält:

```
Output = foo.out
```

```
Environment = i=42; j=23
```

Hiermit kann man Umgebungsvariablen angeben, die der Executable zur Verfügung stehen.

```
Queue
```

Jedes Element, was in die Job-Queue übernommen werden soll, wird hiermit mit allen Parametern, die vorher angegeben wurden, gequeued.

3.2 Aufbau

Der Aufbau einer Submit-Datei:

```
Universe              = vanilla          | Hier sieht man ein Beispiel
Executable            = /usr/bin/R       | einer Submit-Datei.
Getenv                = true             |
Arguments             = –vanilla         | Jede Queue-Anweisung fügt ein
Error                 = x.e              | Element oder so viele, wie
transfer_executable   = false            | dahinter angegeben, zur Queue
Log                   = x.l              | hinzu.
Input                 = x.R              | Variablen werden überschrieben,
                                         | wenn ihnen neue Werte zuge-
Environment           = i=1337           | wiesen werden.
Output                = x1.o             | Somit kann man zum Beispiel
Queue                                    | zwischen zwei Queue-Anweisungen
                                         | die Parameter für die Ausführung
Output                = x2.o             | ändern.
Queue 200                                |
```

Die hier genannten sind nur die wichtigsten, sollten weitere benötigt werden, bitte in dem offiziellen Manual nachschauen.

4 Jobmanagement

4.1 Jobs hinzufügen

```
$ /usr/bin/condor_submit [Submitdatei]
```

Der Beispielbefehl führt dazu, dass die Jobs, die in der Submitdatei bestimmt wurden, abgeschickt wurden. Eine erfolgreiche Rückmeldung der Konsole sieht in etwa so aus:

```
$ /usr/bin/condor_submit Rjobs.submit
Submitting job(s).....................
Logging submit event(s).....................
28 job(s) submitted to cluster 53.
```

4.2 Jobs stoppen

```
# /usr/bin/condor_hold [cluster] | [cluster.process]
```

Das manuelle Stoppen von Jobs führt dazu, dass die Jobs in der Queue als nicht abgearbeitet gelistet bleiben und den Status HELD bekommen. Dies ist beispielsweise dann sinnvoll, wenn ein anderer Job zeitnah bearbeitet werden muss, aber keine Maschine Rechenkapazität frei hat. Ein Beispielaufruf:

```
$ /usr/bin/condor_hold 53
Cluster 53 held.
```

4.3 Gestoppte Jobs wieder starten

```
$ /usr/bin/condor_release [cluster] | [cluster.process]
```

Die gehaltenen Jobs in der Queue bekommen wieder den Status IDLE und werden einer Maschine neu zugeordnet und bearbeitet. Ein Beispielaufruf:

```
$ /usr/bin/condor_release 53
Cluster 53 released.
```

4.4 Jobs entfernen

```
$ /usr/bin/condor_rm [cluster] | [cluster.process]
```

Auf diese Weise können einzelne Jobs durch Angabe der CLUSTER.PROCESS-ID oder eine ganze Reihe von gemeinsam gestarteten Jobs durch Angabe der CLUSTER-ID gelöscht werden. Jeder User kann seine eigenen Jobs verwalten, lediglich privilegierte User können beliebige Jobs entfernen. Ein Beispielaufruf:

```
$ /usr/bin/condor_rm 53
Cluster 53 has been marked for removal.
```

5 Queue verwalten

5.1 Queue anzeigen

```
$ /usr/bin/condor_q
```

In der Liste werden die Jobs, die noch nicht bearbeitet wurden oder gerade bearbeitet werden, angezeigt. Sollte ein Job IDLE sein, wurde er noch nicht an einen EXECUTER übergeben. Hat ein Job den Status HELD, ist er entweder manuell oder auf Grund eines Problems gestoppt worden. Die entsprechenden Logs geben Aufschluss über die Ursache. Wird vor dem Befehl noch ein WATCH mitgegeben, wird die Ausgabe durch automatisches Aktualisieren leichter zu beobachten. Die Ausgabe könnte in etwa so aussehen:[1]

[1] Tabelle stammt aus dem offiziellen Condor Manual

```
$ watch /usr/bin/condor_q
```

ID	OWNER	SUBMITTED	CPU_USAGE	ST	PRI	SIZE	CMD
125.0	jbasney	4/10 15:35	0+00:00:00	I	-10	1.2	hello.remote
127.0	raman	4/11 15:35	0+00:00:00	R	0	1.4	hello
128.0	raman	4/11 15:35	0+00:02:33	I	0	1.4	hello

3 jobs; 2 idle, 1 running, 0 held

5.2 Queueprioritäten verwalten

```
$ /usr/bin/condor_prio [-p priority] [+ | - value] [-n schedd_name]
    [-pool pool_name] cluster | cluster.process | username | -a
```

Die Jobs werden mit Prioritäten versehen, sodass wichtigere Jobs vor unwichtigeren abgearbeitet werden. Beispielaufruf:[2]

```
$ /usr/bin/condor_prio -p -15 126.0
$ /usr/bin/condor_q
```

ID	OWNER	SUBMITTED	CPU_USAGE	ST	PRI	SIZE	CMD
126.0	raman	4/11 15:06	0+00:00:00	I	-15	0.3	hello

1 jobs; 1 idle, 0 running, 0 held

6 Sonstiges

6.1 Verhalten in Ausnahmesituationen

Condor ist sehr robust. Jobs gehen nicht verloren, wenn beispielsweise ein Rechner ausfällt, auf dem Jobs laufen. Die Jobs können, sollte der Rechner im Pool nicht wieder erscheinen, einer anderen Maschine zugewiesen werden. Auf diese Weise entstehen unter Umständen redundate Date, was aber zumindest Datenverlust verhindert.

Grundsätzlich gilt: In der Queue werden die Jobs solange gelistet, bis der MANAGER vom EXECUTER Rückmeldung bekommen hat, dass die Jobs fertig gelaufen sind und die Ergebnisse an den SUBMITTER übergeben wurden. Sollte irgendetwas Unvorhergesehenes passieren, wie der Ausfall einer Maschine, Netzwerkprobleme, etc. werden die Jobs in der Queue gehalten und bekommen den Status HELD. Angehaltene Jobs lassen sich wieder starten und werden beispielsweise einer anderen Maschine zugeordnet.

Das „Worst Case-Szenario“ tritt ein, wenn der MANAGER unwiederbringlich ausfällt. Auf ihm läuft der COLLECTOR, welcher die Informationen über den Pool bereitstellt. Außerdem versuchen die Clients ständig in Kontakt mit dem MANAGER zu sein.

Da die Daten in dem vorliegenden Pool auf einem Fileserver gespeichert werden, würde, im Falle eines Ausfalls des MANAGERS, hauptsächlich Zeit verloren gehen, weil evtl. auch langandauernde Jobs von vorne beginnen müssten. Sollte der Fall eintreten, dass der Fileserver von Condor nicht erreicht wird, ist in diesem Fall das System lahmgelegt, da bei der vorliegenden Konfiguration die Homeverzeichnisse nicht erreichbar sind, aus denen beispielsweise die Jobs abgeschickt werden.

[2] Tabelle stammt aus dem offiziellen Condor Manual

7 Quellen

7.1 Condor Sources

`http://www.cs.wisc.edu/condor/downloads-v2/download.pl` (Stand: 24.02.2016)

7.2 Condor Manual

`http://research.cs.wisc.edu/htcondor/manual/v8.0.5/index.html` (Stand: 24.02.2016)

7.3 PuTTY Download

`http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html` (Stand: 24.02.2016)

7.4 Winscp Download

`http://winscp.net/eng/download.php` (Stand: 24.02.2016)