Obliczenia Wielkiej Skali

Condor

Marcin Rociek
Informatyka, IV rok

1. Wstęp

Condor jest środowiskiem HTC (*High Throughput Computing*), które może zarządzać bardzo dużymi zbiorami rozproszonych stacji roboczych. Środowisko to zbudowane jest w oparciu o architekturę warstwową, która dostarcza potężny i elastyczny zestaw usług zarządzania zasobami (*Resource Management*) dla aplikacji zarówno sekwencyjnych jak i równoległych. Condor jest dostępny dla różnych platform UNIXowych oraz dla Windows NT.

1.1. Siła Condora

Condor w sposób efektywny wykorzystuje moc obliczeniową stacji roboczych komunikujących się poprzez sieć, może również zarządzać dedykowanym klastrem stacji. Jego siła pochodzi ze zdolności do efektywnego podporządkowania istniejących zasobów obliczeniowych pod rozproszone zarządzanie.

Użytkownik przedkłada zadanie do systemu, a Condor szuka maszyny dostępnej w sieci i uruchamia zadanie na tej maszynie. Może wykryć sytuacje, w której maszyna wykonująca zadanie nie jest już dostępna (gdyż być może jej właściciel wrócił z obiadu i zaczął stukać w klawiaturę). Zadanie przechodzi punkt kontrolny i jest przenoszone (migruje) do innej maszyny, która akurat jest dostępna. Condor kontynuuje wykonywanie zadania na innej maszynie dokładnie od punktu, w którym zostało ono przerwane.

Ponieważ Condor może skontrolować i przenieść zadanie, proste staje się zmaksymalizowanie ilości maszyn, które mogą wykonywać zadanie. W tym przypadku nie ma wymagania aby maszyny współdzieliły system plików (np. poprzez NFS lub AFS) a więc wszystkie dostępne w firmie maszyny mogą być wykorzystane do wykonywania zadań, włączając w to maszyny znajdujące się w różnych domenach administracyjnych.

Condor może oszczędzić czas gdy zadanie musi być wykonane wiele (np. kilkaset) razy, być może dla różnych zbiorów danych. Za pomocą jednej komendy wszystkie kilkaset zadań jest przedłożonych Condorowi. Zależnie od ilości maszyn w puli Condora, dziesiątki lub nawet setki wolnych maszyn mogą wykonywać zadanie w danym momencie.

Condor nie wymaga konta na zdalnych maszynach aby wykonać zadanie. Wykorzystuje on bowiem technologię zdalnych wywołań systemowych (*remote system call*), która wyłapuje wywołania systemowych funkcji np. odczytu i zapisu plików na dysk. Wywołania te są transmitowane poprzez sieć aby zostać spełnione na maszynie, na której zadanie zostało przedłożone Condorowi.

Condor dostarcza potężnego zarządzania zasobami poprzez dopasowywanie właścicieli zasobów do konsumentów zasobów. Jest to kamień węgielny dla stworzenia wydajnego środowiska HTC. Przedkładanie zadania Condorowi przez użytkownika dokonywane jest przez tzw. *ClassAd*. Wszyst-

kie maszyny w puli Condora ogłaszają w swojej ofercie (*resource offer*) właściwości swoich zasobów, zarówno statycznych jak i dynamicznych; tj. ilość dostępnej pamięci RAM, typ procesora i jego prędkość, ilość pamięci wirtualnej, lokalizację oraz aktualne obciążenie. Użytkownik określa zaś w ogłoszeniu swoje wymagania zasobów (*resource request*) definiując zbiór zasobów zarówno wymaganych jak i upragnionych do wykonania zadania. Condor spełnia zadanie pośrednika dopasowując ogłoszenia zasobów z wymaganiami zasobów tak aby spełnić wszystkie wymagania użytkownika. Ponadto brane pod uwagę jest kilka warstw priorytetów: priorytet dołączony przez użytkownika do żądania, priorytet użytkownika zgłaszającego oraz wymagania maszyn w puli do akceptowania pewnych typów zgłoszeń ponad innymi.

1.2. Cechy

1.2.1. Punkty kontrolne i migracja

Jeśli program może być zlinkowany z bibliotekami Condora użytkownicy mogą być pewni, że ich zadania zostaną zakończone nawet w ciągle zmieniającym się środowisku wykorzystywanym przez Condora. Jeśli maszyna wykonująca zadanie przestaje być do dyspozycji, zadanie może przejść punkt kontrolny. Zadanie może być kontynuowane po przeniesieniu do innej maszyny. Condor może również okresowo kontrolować zadanie aby zabezpieczyć już wykonaną część zadania przed straceniem w przypadku niepowodzenia, spowodowanym np. wyłączeniem maszyny lub jej awarią.

1.2.2. Zdalne wywołania systemowe

Mimo tego że zadania są uruchamiane na zdalnych maszynach, w trybie *Standard Universe* Condor zachowuje środowisko uruchomienia poprzez zdalne wywołania systemowe. Użytkownicy nie muszą martwić się o tworzenie plików dostępnych zdalnym maszynom, ani nawet o uzyskanie konta na zdalnej maszynie. Program uruchomiony poprzez Condora zachowuje się tak jakby działał na maszynie na której został zgłoszony przez użytkownika, niezależnie od tego na której maszynie faktycznie jest uruchomiony.

1.2.3. Brak konieczności zmian w kodzie źródłowym

Użytkowanie Condora nie wymaga jakichkolwiek zmian w kodzie źródłowym programu. Condor może uruchamiać nie-interaktywne programy. Punkty kontrolne i migracja zadań dokonywana jest przez Condora w sposób przeźroczysty i automatyczny, tak jak i również używanie zdalnych wywołań systemowych. Jeśli tylko te cechy są przez użytkownika pożądane, wystarczy tylko że program zostanie przelinkowany z bibliotekami Condora. Kod źródłowy nie jest ani zmieniany ani nawet przekompilowywany.

1.2.4. Łączenie pul maszyn

Gromadzenie jest cechą Condora która umożliwia uruchamianie zadań zgłoszonych do pierwszej puli maszyn Condora w drugiej puli. Mechanizm ten jest elastyczny, podąża za żądaniami zgłoszonych zadań.

1.2.5. Kolejkowanie zadań

Możliwe jest kolejkowanie wykonywania zadań wymuszone zależnościami pomiędzy zadaniami. Zbiór zadań podawany jest z użyciem skierowanego grafu acyklicznego, w którym każde zadanie jest węzłem grafu. Zadania są przedkładane Condorowi zgodnie z zależnościami podanymi w grafie.

1.2.6. Umożliwienie obliczeń gridowych

Technika *glidein* umożliwia uruchomienie zadań zgłoszonych Condorowi na maszynach gridowych w różnych miejscach na całym świecie.

1.2.7. Wrażliwość na życzenia właścicieli maszyn

Właściciel maszyny ma całkowite pierwszeństwo przed innymi użytkownikami. Właściciel może zezwolić innym na obliczenia kiedy sam nic nie robi, ale chce przejąć pełną kontrolę nad maszyną po powrocie do pracy. Właściciel nie musi wykonywać jakichś czynności aby odzyskać kontrolę, gdyż Condor czyni to automatycznie.

1.2.8. *ClassAd*

Mechanizm *ClassAd* dostarcza bardzo elastyczny i klarowny szkielet dopasowujący żądania zasobów do ofert zasobów. Użytkownicy mogą w łatwy sposób precyzować wymagane zasoby jak również preferowane zasoby. Przykładowo użytkownik może wymagać maszyny z 64MB pamięci RAM ale preferować maszyny z 128MB jeśli tylko taka jest dostępna. Właściciel stacji roboczej może zaś może sobie zażyczyć uruchamiania przez jego stację jedynie zadań należących do określonej grupy użytkowników itp.

1.3. Ograniczenia

1.3.1. Ograniczenie zadań, które mogą być kontrolowane

Chociaż Condor jest w stanie kolejkować i uruchamiać jakikolwiek program, istnieje kilka ograniczeń, które zadania muszą spełniać aby mogły być w sposó przeźroczysty kontrolowane i przenoszone:

1. Niedopuszczalne są zadania wieloprocesowe, włączając w to wywołania funkcji systemowych `fork()`, `exec()` i `system()`.
2. Niedopuszczalna jest komunikacja między procesami za pomocą potoków, semaforów i pamięci dzielonej.
3. Komunikacja sieciowa musi być zwięzła - zadanie *może* utworzyć połączenie sieciowe wykorzystując np. funkcję `socket()`, ale połączenie otwarte przez dłuższy czas spowoduje opóźnienia w kontroli i migracji zadania.
4. Niedopuszczalne jest wysyłanie i odbieranie sygnałów `SIGUSR2` i `SIGTSTP`, które są zarezerwowane wewnętrznie przez Condora. Wysyłanie i odbieranie wszelkich innych sygnałów *jest* dopuszczalne.
5. Niedopuszczalne jest używanie funkcji `alarm()`, `getitimer()` oraz `sleep()`.
6. Wielokrotne wątki na poziomie jądra nie są dopuszczalne. Dopuszczalne są jedynie wątki na poziomie użytkownika.
7. Niedopuszczalne są pliki mapowane w pamięci (`mmap()`, `munmap()`).
8. Blokowanie plików jest dopuszczalne, ale nie jest zachowane pomiędzy kolejnymi kontrolami zadania.
9. Wszystkie pliki mogą być otwarte tylko do odczytu lub tylko do zapisu. Pliki otwarte zarówno do odczytu jak i do zapisu mogą powodować problemy w przypadku gdy zadanie musi być odtworzone do obrazu z ostatniej kontroli.
10. Maszyna, na której przedkładane są zadania musi posiadać dużo wolnej przestrzeni dyskowej na obrazy kontrolne zadań. Wielkość takiego obrazu jest w przybliżeniu równa ilości pamięci wirtualnej zużywanej przez zadanie w czasie pracy. Można zastosować specjalny dodatkowy serwer (*checkpoint server*) przechowujący obrazy kontrolne zadań w puli Condora.

11. W systemach Digital Unix (OSF/1), HP-UX oraz Linux zadanie musi być zlinkowane z bibliotekami statycznie. Linkowanie dynamiczne jest dopuszczalne na wszystkich pozostałych platformach.

1.3.2. Konieczność przelinkowania programów

Choć nie ma konieczności zmian w kodzie i przekompilowywania programów, Condor wymaga aby były one przelinkowane z odpowiednimi bibliotekami aby mogły korzystać z usług kontroli oraz zdalnych wywołań systemowych. Czynnik ten w zasadzie blokuje te usługi dla pakietów komercyjnych, które zwykle nie posiadają udostępnionych skompilowanych plików obiektowych (ani tym bardziej plików źródłowych). Wciąż jednak dla tych pakietów dostępne są wszelkie pozostałe usługi Condora.

1.4. Dostępność

Wersja 6.x dostępna jest dla niżej wymienionych platform:

Architektura	*System operacyjny*
Hewlett Packard PA-RISC (both PA7000 and PA8000 series)	HPUX 10.20
Sun SPARC Sun4m,c, Sun UltraSPARC	Solaris 2.5.x, 2.6, 2.7, 2.8
Silicon Graphics MIPS (R4400, R4600, R8000, R10000)	IRIX 6.5
Intel x86	RedHat Linux 5.2, 6.x, 7.1
	Solaris 2.5.x, 2.6, 2.7
	Windows NT 4.0 („okrojony”)
Digital ALPHA	OSF/1 (Digital Unix) 4.x to 5.0a
	Linux 2.2.x

„Okrojony” oznacza zubożoną wersję Condora, która nie wspiera kontroli zadań ani zdalnych wywołań systemowych. Oznacza to, że możliwa jest jedynie praca w trybie *Vanilla*.

Condor nie jest ani testowany ani nie wspiera innych niż RedHat dystrybucji Linuxa.

2. Użytkowanie

2.1. Co Condor robi?

Condor jest specjalizowanym systemem wsadowym służącym do zarządzania intensywnych obliczeniowo zadań. Jak większość systemów wsadowych, Condor dostarcza mechanizm kolejkowania, politykę szeregowania zadań, zasady priorytetowania zadań oraz klasyfikację zasobów. Użytkownik przedkłada zadanie Condorowi, który to zadanie umieszcza w kolejce, uruchamia a potem informuje użytkownika o wyniku.

Condor może skutecznie wykorzystywać zarówno maszyny dedykowane do obliczeń, jak i każde inne nie-dedykowane maszyny, które aktualnie nie są używane (brak aktywności klawiatury, brak obciążenia procesora, brak aktywnych zdalnych użytkowników itp). Condor potrafi efektywnie zaprząc do pracy te maszyny, które w przeciwnym wypadku stały by po prostu bezczynnie.

2.2. Środowiska

Zadanie może pracować w jednym z czterech dostępnych trybów pracy (*Condor Universe*):

- Standard
- Vanilla
- PVM
- Globus

2.2.1. *Standard Universe*

Środowisko *Standard Universe* umożliwia stosowanie kontrolowania zadań oraz używanie zdalnych wywołań systemowych. Cechy te powodują zwiększenie niezawodności danego zadania oraz umożliwiaja jodnolity dostęp do zasobów z każdej maszyny w puli wszystkich maszyn. Aby program mógł być wykonywany w środowisku *Standard* musi on być przelinkowany z odpowiednimi bibliotekami Condora za pomocą narzędzia *condor_compile.* Większość programów może być przygotowanych do pracy w tym środowisku, istnieje jednak kila ograniczeń.

Condor przeprowadza kontrolę zadania w równych odstępach czasu. Obraz z kontroli jest po prostu odzwierciedleniem stanu programu w danym momencie. Jeśli zadanie musi zostać przeniesione na inną maszynę, Condor tworzy obraz zadania, kopiuje go na nową maszynę i uruchamia dokładnie od tego punktu, w którym zakończyło pracę. W przypadku awarii maszyny Condor odtwarza zadanie na nowej maszynie używając do tego celu najświeższego obrazu jaki posiada.

2.2.2. *Vanilla Universe*

Środowisko *Vanilla* przeznaczone jest dla programów, które nie mogą być przelinkowane z bibliotekami Condora (gdyż np. nie ma dostępu do plików obiektów, a jedynie do wersji binarnej). Z tego też względu programy nie mogą w tym środowisku opierać się o mechanizm kontroli zadań i zdalne wywołaniania systemowe. Jeśli zadanie jest tylko częściowo wykonane na jakiejś maszynie to Condor ma w tej sytuacji tylko dwa wyjścia: wstrzymać wykonywanie zadania i czekać na ponowne zwolnienie maszyny lub poddać się i uruchomić zadanie od nowa na innej maszynie.

2.2.3. *PVM*

Środowisko *PVM* umożliwia wykorzystanie Condora przez programy używające interfejsu PVM (*Parallel Virtual Machine*).

2.2.4. *Globus Universe*

Środowisko *Globus Universe* przeznaczone jest dla zadań przeznaczonych dla systemu Globus. Każde kolejkowane zadanie jest tłumaczone na odpowiedni Globusowy skrypt RSL i przedkładane programowi *globusrun.*

2.3. *Personal Condor*

Pod pojęciem *Personal Condor* należy rozumieć Condora pracującego na pojedynczej maszynie, nie wymagającego ani dostępu do roota ani interwencji administratora. Wbrew pozorom już na jednej maszynie można z pożytkiem korzystać z usług Condora, który będzie:

- opiekował się zleconymi zadaniami i informował o ich postępie;
- stosował się do ustalonych preferencji co do kolejności wykonywania zadań;
- tworzył odpowiednie logi zadań;
- uodparniał zadania na awarie systemu;
- stosował się do ustalonych reguł na temat czasu, w którym zadania mogą być uruchamiane.

2.4. Opis zadania

Aby przedłożyć zadanie Condorowi należy wybrać odpowiednie środowisko pracy (np. *Vanilla* lub *Standard*). Aby program mógł być uruchomiony przez Condora, musi być napisany tak, aby mógł być przetwarzany wsadowo, tj. bez konieczności jakiejkolwiek interakcji, bez żadnego GUI itp. Może oczywiście wykorzystywać strumienie `STDIN`, `STDOUT` i `STDERR`, lecz zamiast odpowiednich urządzeń wejściowych i wyjściowych będą do tego celu wykorzystane po prostu pliki.

Następnie należy stworzyć odpowiedni plik opisujący (SDT - *Submit Description File*), który jest zwykłym plikiem tekstowym przekazującym Condorowi informacje o pliku wykonywalnym, środowisku, wejściu i wyjściu, parametrach programu, zmiennych środowiskowych oraz specjalnych wymagań i preferencji. Plik taki może od razu definiować więcej uruchomień tego programu, z których każde może posiadać inne argumenty wywołania, inne wejście, wyjście itp.

Przykład pliku opisującego, w którym precyzujemy środowisko, nazwę pliku wykonywalnego, pliki wejściowe i wyjściowe, argumenty oraz katalog, w którym będą umieszczone pliki wyjściowe (w którym powinny się również znajdować pliki wejściowe):

```
Universe        = standard
Executable      = test
Input           = test.stdin
Output          = test.stdout
Error           = test.stderr
Arguments       = -arg1 -arg2
InitialDir      = run_1
queue
```

Podanie nazw katalogów umożliwia za pomocą jednego pliku opisującego uruchomić ten sam program dla różnych danych wejściowych. Pliki wynikowe znajdą się wtedy w odpowiednich podkatalogach:

```
Universe        = standard
Executable      = test
Arguments       = -arg1 -arg2
InitialDir      = run_0
queue
InitialDir      = run_1
queue
```

Z pomocą makra `$(Process)` można za jednym zamachem zakolejkować dużą ilość uruchomień:

```
Universe        = standard
Executable      = test
Arguments       = -arg1 -arg2
InitialDir      = run_$(Process)
queue 100
```

Taka konfiguracja spowoduje 100-krotne uruchomienie programu, a dane wyjściowe zostaną umieszczone kolejno w katalogach `run_0`, `run_1`, ..., `run_99`.

Oprócz podstawowych informacji na temat uruchamianego programu, w pliku opisującym można zawrzeć np. wymagania tego programu co do zasobów. Przykładowo aby Condor uruchamiał zadanie tylko na maszynach mających więcej niż 64MB pamięci RAM, wystarczy do pliku opisującego dodać linię:

```
Requirements    = Memory >= 64
```

2.5. Stan dostępnych maszyn

Do sprawdzania stanu maszyn w puli służy komenda *condor_status*. Bez dodatkowych argumentów wyświetla ona skrótowe informacje na temat poszczególnych maszyn, m.in. aktualne obciążenie procesora, ilość pamięci oraz stan (z punktu widzenia Condora). Maszyny znajdujące się w stanie `Unclaimed` są gotowe do wykonywania zadań Condora. Jak widać na przykładowym wydruku niedostępna jest maszyna `ant2`, która jest obciążona w 100% oraz `anthill`, który mimo małego obciążenia jest niedostępny ze względu na pracujących na nim (zdalnie) użytkowników.

```
[mrock@anthill condor]$ condor_status

Name          OpSys       Arch   State      Activity   LoadAv Mem   ActvtyTime

ant1.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:12:14
ant2.cluster  LINUX       INTEL  Owner      Idle       1.000    60  1+02:23:23
ant3.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:08:58
ant4.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:01:59
ant5.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:09:11
ant6.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:09:07
ant7.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:09:05
ant8.cluster  LINUX       INTEL  Unclaimed  Idle       0.000    60  0+02:09:09
anthill.ki.ag LINUX       INTEL  Owner      Idle       0.170   249  0+04:10:04

                     Machines Owner Claimed Unclaimed Matched Preempting

         INTEL/LINUX        9     2       0         7       0          0

               Total        9     2       0         7       0          0
```

Szczegółowe informacje na temat maszyny `ant3`:

```
[mrock@anthill condor]$ condor_status -l ant3.cluster
MyType = "Machine"
TargetType = "Job"
Name = "ant3.cluster"
Machine = "ant3.cluster"
Rank = 0.000000
CpuBusy = ((LoadAvg - CondorLoadAvg) >= 0.500000)
CondorVersion = "$CondorVersion: 6.2.0 Mar  8 2001 $"
CondorPlatform = "$CondorPlatform: INTEL-LINUX-GLIBC20 $"
VirtualMachineID = 1
VirtualMemory = 262504
Disk = 2497580
CondorLoadAvg = 0.000000
LoadAvg = 0.000000
KeyboardIdle = 95052
ConsoleIdle = 36035391
Memory = 60
Cpus = 1
StartdIpAddr = "<192.168.0.13:3006>"
Arch = "INTEL"
OpSys = "LINUX"
UidDomain = "ki.agh.edu.pl"
FileSystemDomain = "ant3.cluster"
Subnet = "192.168.0"
TotalVirtualMemory = 262504
TotalDisk = 2497580
KFlops = 132989
```

```
Mips = 656
LastBenchmark = 1018209614
TotalLoadAvg = 0.000000
TotalCondorLoadAvg = 0.000000
ClockMin = 20
ClockDay = 1
TotalVirtualMachines = 1
CpuBusyTime = 0
CpuIsBusy = FALSE
State = "Unclaimed"
EnteredCurrentState = 1018209980
Activity = "Idle"
EnteredCurrentActivity = 1018209980
Start = ((LoadAvg - CondorLoadAvg) <= 0.300000) && KeyboardIdle > 15 * 60
Requirements = START
CurrentRank = -1.000000
LastHeardFrom = 1018218018
```

Jak widać z parametru `Start`, jest ona skonfigurowana tak, że Condor uruchamia na niej programy tylko wtedy, gdy obciążenie procesora jest mniejsze niż 30% i czas bezczynności jest większy niż 15 minut.

2.6. Uruchamianie zadania

Tworzymy przykładowy program testujący oraz minimalny plik opisujący `test.cmd`:

```
executable = test
log = test.log
queue
```

Zadanie zostaje przedłożone Condorowi:

```
[mrock@anthill condor]$ condor_submit test.cmd
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 169.
```

Po zakończeniu działania można sprawdzić stworzony przez Condora log, który zawiera m.in. informacje na temat maszyny z której wysłano zadanie, maszyny która wykonała zadanie, czasu działania programu itp.:

```
[mrock@anthill condor]$ cat test.log
000 (169.000.000) 04/08 00:22:28 Job submitted from host: <149.156.100.60:42956>
...
001 (169.000.000) 04/08 00:22:31 Job executing on host: <192.168.0.14:4034>
...
005 (169.000.000) 04/08 00:22:31 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        224  -  Run Bytes Sent By Job
        14423  -  Run Bytes Received By Job
        224  -  Total Bytes Sent By Job
        14423  -  Total Bytes Received By Job
...
```

2.7. Uruchamianie wielu zadań

Ciekawiej sprawa wygląda, gdy poprosimy Condora o 20-krotne uruchomienie programu testującego modyfikując plik opisujący zadanie:

```
executable = test
log = test.log
queue 20
```

Przedkładamy Condorowi zadanie do uruchomienia:

```
[mrock@anthill condor]$ condor_submit test.cmd
Submitting job(s)...................
Logging submit event(s)...................
20 job(s) submitted to cluster 170.
```

2.8. Sprawdzanie kolejki zadań

Do sprawdzenia zawartości kolejki służy polecenie *condor_q*. Tuż po wydaniu polecenia *condor_submit* kolejka zawiera 20 zadań, z których 7 jest właśnie wykonywane (na 7 dostępnych maszynach). Pozostałe 13 oczekuje na możliwość uruchomienia.

```
[mrock@anthill condor]$ condor_q

-- Submitter: anthill.ki.agh.edu.pl : <149.156.100.60:42956> : anthill.ki.agh.edu.pl
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 170.0   mrock           4/8  00:25   0+00:00:01 R  0   0.0  test
 170.1   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.2   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.3   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.4   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.5   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.6   mrock           4/8  00:25   0+00:00:00 R  0   0.0  test
 170.7   mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.8   mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.9   mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.10  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.11  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.12  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.13  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.14  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.15  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.16  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.17  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.18  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test
 170.19  mrock           4/8  00:25   0+00:00:00 I  0   0.0  test

20 jobs; 13 idle, 7 running, 0 held
```

3. Administracja

Plik `condor_config` jest globalnym plikiem konfiguracyjnym dla wszystkich dostępnych maszyn. Zawiera on między innymi reguły określające kiedy Condor może na jakiejś maszynie zadanie uruchomić, kiedy ma je przerwać, kontynuować lub usunąć. Domyślne ustawienie jest takie, że maszyna jest gotowa do wykonania zadania Condora jeśli jest obciążona w mniej niż 30% i bezczynna przez dłużej niż 15 minut. Język opisujący te reguły jest jednak na tyle elastyczny, że możliwe jest

utworzenie praktycznie dowolnych warunków, np. zezwolanie Condorowi na pracę tylko w nocy, preferowanie zadań pewnych użytkowników itp.

Do zmiany konfiguracji 'w locie' służy polecenie *condor_reconfig*, które uruchamiamy po wprowadzeniu odpowiednich zmian w pliku `condor_config`:

```
[condor@anthill sbin]$ ./condor_reconfig ‘condor_status -master‘
Sent "Reconfig" command to master ant1.cluster
Sent "Reconfig" command to master ant2.cluster
Sent "Reconfig" command to master ant3.cluster
Sent "Reconfig" command to master ant4.cluster
Sent "Reconfig" command to master ant5.cluster
Sent "Reconfig" command to master ant6.cluster
Sent "Reconfig" command to master ant7.cluster
Sent "Reconfig" command to master ant8.cluster
Sent "Reconfig" command to master anthill.ki.agh.edu.pl
```

Standardowo instalacja Condora zakłada, że wszystkie daemony pracują z prawami roota. Pozwala to zadaniom uruchamianym przez Condora na dostęp do wszystkich plików dostępnych użytkownikowi przedkładającemu zadanie. Jeśli daemony nie pracują z prawami roota, użytkownik musi udostępnić do odczytu i/lub zapisu część lub nawet wszystkie pliki i katalogi używane przez uruchamiane zadanie użytkownikowi o `UID` równym parametrowi `RealUid`. W większości przypadków wymusza to konieczność wykonania polecenia *chmod 777* na katalogu, w którym znajduje się zadanie przeznaczone dla Condora.

4. Polecenia

Krótki opis poleceń Condora:

condor_checkpoint	skontrolowanie zadań pracujących na wyspecyfikowanych maszynach
condor_compile	stworzenie pliku wykonywalnego zlinkowanego z bibliotekami Condora
condor_config_val	odczytanie lub ustawienie zmiennej konfiguracyjnej Condora
condor_findhost	wyszukanie maszyny spełniającej podane ograniczenia, której odjęcie z puli maszyn będzie miało minimalny wpływ na już pracujące zadania
condor_glidein	dodanie zasobu Globusa do puli Condora
condor_history	odczyt logów zakończonych zadań Condora
condor_hold	wprowadzenie zadań z kolejki w stan zatrzymania (*hold*)
condor_master	główny daemon Condora
condor_off	wyłączanie daemonów Condora
condor_on	włączenie daemonów Condora
condor_preen	usunięcie niepotrzebnych plików z katalogów Condora
condor_prio	zmiana priorytetów zadań w kolejce
condor_q	wyświetlenie informacji o zadaniach w kolejce
condor_qedit	modyfikacja parametrów zadania
condor_reconfig	zmiana konfiguracji daemonów Condora
condor_release	uwolnienie zatrzymanych zadań w kolejce
condor_reschedule	uaktualnienie informacji o szeregowaniu dla centralnego zarządcy

condor_restart	zrestartowanie któregoś z daemonów Condora (domyślnie *condor_master*)
condor_rm	usunięcie zadań z kolejki
condor_run	przedłożenie komendy powłoki jako zadania Condora
condor_stats	wyświetlenie statystyk na temat puli Condora
condor_status	wyświetlenie stanu puli Condora
condor_submit	skolejkowanie zadań na zdalnych maszynach
condor_submit_dag	zarządzanie i kolejkowanie zadań z wyspecyfikowanym skierowanym grafem acyklicznym do wykonania na zdalnych maszynach
condor_userlog	wyświetlenie statystyk zadań na podstawie ich logów
condor_userprio	zarządzanie priorytetami użytkowników
condor_vacate	porzucenie zadań uruchomionych na zdalnych maszynach

Bibliografia

[1] Condor Team, University of Wisconsin-Madison; *Condor Version 6.3.1 Manual*. `http://www.cs.wisc.edu/condor/manual/v6.3/index.htm`

[2] Peter Couvares and Todd Tannenbaum; *Condor Tutorial, First EuroGlobus Workshop, June 2001*. `http://www.cs.wisc.edu/condor/slides/euroglobus-tutorial/euroglobus-tutorial.html` `http://www.cs.wisc.edu/condor/slides/euroglobus-tutorial/euroglobus-tutorial.ppt`