Dostęp do konta oraz logowanie

Wstęp

Warunki korzystania z zasobów obliczeniowych klastra CIŚ:

- Należy posiadać certyfikat (składanie wniosków o wydanie certyfikatu na https:/ca.cis.gov.pl/)
- Poprawnie skonfigurowany klient VPN (np. OpenVPN);
- Posiadanie konta w systemie klastra (LDAP).

Dostęp do konta

Dostęp do konta użytkownika oraz zawartych na nich danych możliwy jest (po uprzednim spełnieniu warunków korzystania z klastra CIS) poprzez serwer dostępowy udostępniany w systemie klastra.

Parametry serwera dostępowego (wewnątrz sieci VPN):

- Nazwa serwera: **labusrint.cis.gov.pl;**
- Adres IP serwera: **10.200.2.198;**
- Dostępne protokoły: **SSH (port 22).**

labusrint.cis.gov.pl

Na maszynę **labusrint** logujemy się po zestawieniu połączenia VPN poleceniem (dla systemów Linux):

```
$ ssh login@labusrint.cis.gov.pl
```

Dla systemu Windows skorzystać należy natomiast z dodatkowego oprogramowania (np. Putty) podając w konfiguracji parametry serwera dostępowego.

Katalog domowy użytkownika na labusrint jest katalogiem sieciowym dzielonym z węzłami obliczeniowymi na klastrze. W tym katalogu należy umieszczać skrypty do których mają mieć dostęp zadania na klastrze jak i jest to miejsce gdzie zadania powinny zapisywać wyniki obliczeń.

Zasoby obliczeniowe

Udostępniane zasoby

Zasoby obliczeniowe klastra dostępne są w ramach systemu kolejkowego pracującego pod kontrolą oprogramowania Torque + MAUI. Użytkownicy mogą uruchamiać skrypty/aplikacje obliczeniowe z poziomu maszyn dostępowych, natomiast samo wykonanie aplikacji odbywa się w postaci zadań pracujących bezpośrednio na przydzielonych węzłach obliczeniowych.

Struktura klastra

System klastra oparty jest o następującą strukturę logiczną:

- Serwer dostępowy: labusrint.cis.gov.pl – jest to jedyny serwer na, który Użytkownicy powinni logować się bezpośrednio;
- Serwer kolejkowy: labqsrv.cis.gov.pl – nadzoruje pracę systemu kolejkowego;
- Węzły obliczeniowe: labwn0x.cis.gov.pl – węzły obliczeniowe.

W ramach klastra obliczeniowego udostępniane są serwery obliczeniowe o następujących parametrach:

Informacje o sprzęcie:	
• Model serwera:	SuperBlade TwinBlade [LINK]
• Przeznaczenie:	Węzeł obliczeniowy
• Zakres nazw:	labwn01.cis.gov.pl – labwn03.cis.gov.pl
• Dostępna ilość maszyn:	3
• Adresacja:	10.200.2.1 - 10.200.2.3
• Dostępność:	Wewnątrz sieci VPN
Informacje o CPU:	
• Model CPU:	Intel Xeon E5-2680 v2 [SPECYFIKACJA]
• Ilość CPU / serwer:	2
• Ilość rdzeni fizycznych / CPU:	10
• Ilość wątków / rdzeń CPU:	2
• Całkowita ilość rdzeni / serwer:	20
• Całkowita ilość wątków / serwer:	40
• Taktowanie CPU (nominalne/tryb TurboBoost):	2800Mhz / 3600Mhz
Pamięć RAM:	
• Dostępna pamięć RAM:	128 GB
• Rodzaj:	DDR3
Sieć:	
• Interconnect:	1 x Ethernet (10Gbit/sek per port) 1 x Infiniband (40Gbit/sek per port)
Systemy plików:	

• Partycja /scratch:	• Lokalny system plików (nie współdzielony); • Dostępna pojemność / serwer: ~250 GB; • System plików: (ext4) over SATA • Wydajność: do 500MB/sek (odczyt); do 200MB/sek (zapis)
• Zasób /mnt/home:	• System plików współdzielony; • Interconnect: Ethernet 10Gbit/sek; • Przepustowość: do 1GB/sek;
Benchmarki:	
• Wydajność teoretyczna:	448 GFLOPS

Uruchamianie zadań

Poniżej znajduje się opis uruchamiania różnych typów zadań za pomocą systemu kolejkowego Torque.

Ściągawka - popularne polecenia Torque

Nazwa polecenia	Zastosowanie
qsub	Submitowanie zadań
qdel	Usuwanie zadań
qstat	Wyświetlenie raportu nt. aktualnego stanu zadań
qnodes	Wyświetlenie raportu nt. aktualnego stanu węzłów obliczeniowych

Zadania sekwencyjne

Zadania sekwencyjne to najprostszy typ zadań - działają one w trybie wsadowym w obrębie pojedynczej maszyny obliczeniowej.

Do submitowania służy komenda **qsub**.

Musimy jej dostarczyć parametry, w najprostszym przypadku wystarczy jeden: nazwa skryptu z zadaniem. Dodatkowo można dostarczyć (za pomocą przełącznika *-q*) nazwę kolejki w obrębie której ma zostać uruchomione zadanie (obecnie jedyną dostępną kolejką jest kolejka o nazwie **qlab**). W przypadku braku tego argumentu zostanie wybrana kolejka domyślna.

Przykładowe wywołanie polecenia *qsub*:

```
qsub test.ah            #Uruchamiany skrypt o nazwie 'test.sh'
qsub -q qlab test.sh    #Uruchamiany skrypt o nazwie 'test.sh'
qsub -I                 #Zadanie interaktywne/kolejka oraz zasoby domyślne
```

Definiowanie zasobów sprzętowych:

Podczas wysyłania zadania (polecenie *qsub*) istnieje możliwość zadeklarowania ilości węzłów obliczeniowych/rdzeni, które mają zostać przydzielone do zadania. Definicję zasobów wykonujemy w następujący sposób:

```
qsub -l nodes=<ilość_węzłów>:ppn=<ilość_rdzeni_na_węzeł> ...
```

gdzie:

- <ilość_węzłów> - ilość osobnych maszyn obliczeniowych, które mają być zarezerwowane na potrzeb zadania;
- <ilość_rdzeni_na_węzeł> - ilość rdzeni przydzielona do zadania w obrębie pojedynczego węzła obliczeniowego;

Całkowita ilość rdzeni CPU przypisana do zadania wynosi <ilość_węzłów>*<ilość_rdzeni_na_węzeł>.

Należy mieć na uwadze, aby przy deklaracji każdego z podanych parametrów nie przekroczyć ich maksymalnej liczby (dane podane w tabelce z kolejkami oraz w specyfikacji technicznej maszyn klastra). Podanie zbyt dużych wartości (ilość węzłów > całkowitej ilości węzłów danego typu jaką dysponuje klaster, lub ilość rdzeni na węzeł > całkowitej ilości rdzeni jaką dany węzeł dysponuje) skutkować będzie niemożnością uruchomienia się danego zadania. Błędne deklaracje (np. zawyżone wartości) nie będę zgłaszane przez system, gdyż z punktu widzenia parametrów dla zadania są one poprawne składniowo, nie spełniają jednak specyfikacji klastra obliczeniowego.

Przykłady definiowania zasobów sprzętowych (ilość węzłów/CPU):

```
qsub -l nodes=1:ppn=4 test.sh    #1 węzeł obliczeniowy/4 rdzenie
qsub -l nodes=1:ppn=1 test.sh    #1 węzeł obliczeniowy/1 rdzeń
qsub -l nodes=1:ppn=40 test.sh    #1 węzeł obliczeniowy/40 rdzeni - rezerwacja całego węzła
```

Przykłady niepoprawnych zestawów parametrów dla qsub - **te zadania nigdy się nie uruchomią**:

```
qsub -l nodes=1:ppn=64 test.sh    #Źle: Zbyt duża ilość rdzeni (procesory Intel posiadają ich max 40)
qsub -q qtest -l nodes=1:ppn=1 test.sh        #Źle: Nieprawidłowa nazwa kolejki, kolejka 'qtest' nie istnieje
qsub -l nodes=40:ppn=1 test.sh    #Prawdopodobnie źle: zamieniona miejscami ilość węzłów/rdzeni
```

W tym wypadku chcemy uruchomić skrypt test.sh. Skrypt test.sh jest zwykłym skryptem basha i w moim przypadku zawiera tylko:

```
#!/bin/bash
/bin/hostname
```

Komenda qsub powinna nam zwrócić identyfikator zadania, na przykład:

```
27.qsrv.cis.gov.pl
```

Możemy sprawdzić status naszego zadania wpisując:

```
qstat
```

W ten sposób dowiemy się czy zadanie czeka w kolejce czy się liczy (zadania oczekujące mają status ‘Q’, natomiast uruchomione status ‘R’).

Polecenie:

```
qstat -f <id_zadania>
```

…wyświetli rozszerzone informacje o naszym zadaniu.

Po zakończeniu zadania pojawią się w naszym katalogu domowym dwa pliki (ścieżka taka z jakiej zostało wykonane polecenie qsub):

```
test.sh.o27
test.sh.e27
```

Pierwszy zawiera standardowe wyjście (STDOUT), drugi standardowe wyjście błędów (STDERR). Jeżeli przed ukończeniem zadania spostrzegliśmy że jednak jest ono błędne i chcemy je anulować aby zaoszczędzić zasoby dla innych, używamy komendy

```
qdel 27.qsrv.cis.gov.pl
```

I to w zasadzie tyle. Katalog $HOME jest wspólny dla usrint i klastra więc z zadania można zapisywać w nim też inne pliki. Katalog tymczasowy przypisany dla danego zadania zdefiniowany jest w zmiennej $TMPDIR. Znajduje się on na lokalnym dysku maszyny zapewniając szybki zapis i odczyt, należy pamiętać o skopiowaniu wyników na dysk dzielony np do $HOME.

Dokładną listę opcji qsub i qstat można znaleźć w manualu.

Zadania interaktywne

Zadania interaktywne pozwalają na "zalogowanie się" na węzeł obliczeniowy. Pozwala to na odciążenie maszyny User Interface gdyż użytkownik może wykonywać pracę interaktywną o dużych potrzebach obliczeniowych bezpośrednio na klastrze. Przykładowymi zastosowaniami są: kompilacja i testowanie oprogramowania, testowanie skryptów obliczeniowych, etc.

Zadania interaktywne submituje się analogicznie do zadań sekwencyjnych czy też MPI. W wywołaniu komendy qsub zastępujemy nazwę skryptu poprzez parametr "-I"

Zadanie z wykorzystaniem 1 procesora:

```
qsub -I
```

Zadanie z wykorzystaniem kilku procesorów (np: do testów MPI, MP, kompilacji równoległej typu "make -j", etc):

```
qsub -l nodes=1:ppn=4 -I
```

Parametry polecenia "qsub"

W przypadku polecenia "qsub" można wykorzystać następujące parametry:

I - interactive

```
labusrint$ qsub -I
```

Uruchamia zadanie w sposób interaktywny.

q - wybór kolejki

```
labusrint$ qsub -q [nazwa_kolejki]
```

Przypisuje zadanie do określonej kolejki (obecnie jedyną dostępną jest kolejka o nazwie **qlab**).

l - definicja zasobów

```
labusrint$ qsub -l nodes=1:ppn=32
```

Określa ilość maszyn (nodes), które chcemy użyć oraz ilość rdzeni (ppn) na każdej z nich.

```
labusrint$ qsub -l host=labwn01.cis.gov.pl
```

Pozwala "dostać się" na konkretną maszynę obliczeniową.

v/V - eksport zmiennych środowiskowych

```
labusrint$ qsub -v [lista zmiennych]
```

Pozwala na zadeklarowanie własnych zmiennych (w linii poleceń) wraz z ich wartościami, które zostaną wyeksportowane na maszynę docelową jako zmienne środowiskowe.

```
labusrint$ qsub -V
```

Wymusza eksport wszystkich zmiennych środowiskowych na maszynę docelową.

t - zadania macierzowe

```
labusrint$ qsub -t [zakres]
```

Wykonuje wiele zadań jednocześnie. Parametr "zakres" może zawierać kombinację w postaci "A,B,C,D-F,G-H,..." gdzie A,B,C,... to liczby definiujące indeksy uruchomionych zadań.

k - przekierowanie stdout/stderr

```
labusrint$ qsub -k o
labusrint$ qsub -k e
labusrint$ qsub -k eo
```

Dodanie tego przełącznika powoduje przekierowanie strumieni wyjściowych uruchomionego procesu (odpowiednio: **stdout** (dla opcji "o"), **stderr** (dla opcji "e"), **obydwóch** (dla opcji "eo" lub "oe")) bezpośrednio do plików znajdujących się w katalogu domowym użytkownika (/mnt/home/<nazwa_użytkownika>). W normalnej sytuacji pliki wyjścia tworzone są na węźle obliczeniowym i kopiowane do katalogu domowego (**w miejsce położenia**

wykonywanego skryptu) dopiero po zakończenia się zadania, natomiast przy zastosowaniu opcji "-k" użytkownik jest w stanie na bieżąco śledzić strumienie wyjściowe procesu (pliki wyjściowe generowane są wtedy **bezpośrednio w katalogu domowym**, bez względu na położenie skryptu zadania).

a - określenie czasu oczekiwania

```
labusrint$ qsub -a <data>
```

Umożliwia podanie dokładnej daty/godziny (w postaci [[[[CC]YY]MM]DD]hhmm[.SS]), po której zadanie może przejść w stan wykonywania. Przed upływem podanej godziny zadanie będzie w stanie "Waiting".

Zmienne środowiskowe w zadaniach obliczeniowych

Zmienne środowiskowe eksportowane przez system Torque

Opis ważniejszych zmiennych środowiskowych eksportowanych dla uruchomionego zadania przez system Torque. Zmienne te dostępne są z poziomu uruchomionego zadania.

- **PBS_O_WORKDIR**

Zmienna ta wskazuje na katalog domowy dla danego zadania. Zwykle jest to katalog, z którego wykonano polecenie "qsub". Ścieżka jest w postaci bezwzględnej.

Przykład użycia:

```
mkdir $PBS_O_WORKDIR/test       #Tworzy katalog o nazwie 'test' w katalogu domowym dla uruchomionego zadania.
```

- **PBS_ARRAYID**

Zmienna dostępna tylko w obrębie zadań tablicowych. Zawiera ona numer indeksu zadania wewnątrz tablicy zadań. Typowe zastosowania to: rozróżnienie poszczególnych pod-zadań wewnątrz wspólnego skryptu, uzależnienie inputu (np. seed'a do generatora wartości pseudolosowych), etc.

Wartości zmiennej odpowiadają dokładnie indeksom dla pod-zadań. Np.:

Zadanie uruchomione w ten sposób:

```
qsub -t 0-3 test.sh
```

...będzie wykonane w postaci 4 pod-zadań. W każdym z pod-zadań wartość w.w. zmiennej będzie wynosiła odpowiednio od 0 do 3.

- **PBS_ENVIRONMENT**

Określa w jaki sposób zostało uruchomione zadanie.

Typowe wartości

```
PBS_INTERACTIVE     #Zadanie zostało uruchomione w trybie interaktywnym
PBS_BATCH           #Zadanie zostało uruchomione w trybie wsadowym
```

- **TMPDIR**

Zmienna ta zawiera ścieżkę do katalogu tymczasowego dla zadania. Katalog taki, tworzony jest na partycji /scratch w momencie uruchomienia zadania i wykorzystywany jest przez klienta Torque do przechowywania plików tymczasowych zadania (wyjście stdout oraz stderr). Użytkownik może w tym katalogu umieszczać swoje własne pliki tymczasowe na potrzeby uruchomionych procesów.

Należy mieć na uwadze, że:

1. Cała zawartość katalogu tymczasowego zadania zostanie bezpowrotnie usunięta w momencie zakończenia się/usunięcia w.w. zadania w systemu kolejkowego;
2. Ponieważ partycja /scratch nie jest współdzielonym systemem plików, zadanie wykorzystujące wiele węzłów, będzie miało przypisane wiele katalogów tymczasowych (na każdym z węzłów zmienna ta będzie wskazywała na lokalną partycję dla danego węzła obliczeniowego).

Zawartość zmiennej TMPDIR ma zwykle postać:

```
/scratch/<ID_ZADANIA>.labqsrv.cis.gov.pl
```

Zadania OpenMP

Kompilacja

Kompilację wykonujemy z poziomu węzłów obliczeniowych. Najprościej wykonać to poprzez uzyskanie dostępu do linii poleceń węzła obliczeniowego. W tym celu należy uruchomić zadanie interaktywne:

```
labusrint$ qsub –I
```

Po uzyskaniu dostępu do węzła obliczeniowego, należy uruchomić kompilację wywołując kompilator **gcc** z odpowiednimi przełącznikami. W przypadku kodu testowego, linijka uruchamiająca kompilację będzie wyglądała następująco:

```
labwn01$ gcc -fopenmp -o hello_openmp.bin hello_openmp.c
```

W wyniku kompilacji zostanie utworzony wykonywalny plik o nazwie **hello_openmp.bin**.

Uruchamianie

Skompilowane aplikacje OpenMP korzystają ze zmiennej środowiskowej o nazwie **OMP_NUM_THREADS**. Określa ona na ile wątków zostaną zrównoleglone oznaczone fragmenty kodu. Brak tej wartości w puli zmiennych środowiskowych powłoki z której zostanie wykonany kod OpenMP będzie skutkował użyciem tylko jednego wątku.

Zmienną środowiskową można wyeksportować poleceniem:

```
labwn01$ export OMP_NUM_THREADS=<wartość>
```

Usuwanie wartości z puli zmiennych środowiskowych poleceniem:

```
labwn01$ unset OMP_NUM_THREADS
```

Natomiast samo uruchomienie aplikacji, przy pomocy polecenia (dla skompilowanego w przykładzie kodu testowego):

```
labwn01$ ./hello_openmp.bin
```

Uruchamianie zadań OpenMP z wykorzystaniem systemu kolejkowego

W celu ułatwienia sobie uruchamiania kodu OpenMP (np. w przypadku testowania aplikacji) można utworzyć skrypt, który wysłany przy pomocy systemu kolejkowego na węzeł obliczeniowy, uruchomi nasze zadanie a następnie zwróci nam wyniki jego pracy.

Dla przykładowego kodu może on mieć następującą postać:

```
#!/bin/bash
#Czy jestesmy w obrebie zadania obliczeniowego?
if [[ -z "$PBS_JOBID" ]]; then
        echo "WARNING: Not in PBS job. Exiting."
        echo "Please run: qsub $(basename $0)"
        exit 1
fi
#Uruchomienie kompilacji
cd $PBS_O_WORKDIR
gcc -fopenmp -o hello_openmp.bin hello_openmp.c
#Czy kompilacja przebiegla pomyslnie?
if [[ $? -eq 0 ]]; then
        #Ustawienie zmiennej srodowiskowej OMP_NUM_THREADS
        #rowna ilosci rdzeni przypisanej do zadania
        export OMP_NUM_THREADS=$PBS_NUM_PPN
        #Uruchomienie skompilowanego kodu
        ./hello_openmp.bin
else
        #Informacja o bledach
        echo "Cannot run the code. There were some compilation errors."
        exit 2
fi
```

Tak przygotowany kod można uruchomić poleceniem:

```
labusrint$ qsub -l nodes=1:ppn=10 run.sh
```

W tym przypadku kod zostanie wykonany przy użyciu 10 wątków. Ilość wątków można sprecyzować podczas wysyłania zadania jako parametr polecenia „qsub”.

Po wykonaniu się zadania obliczeniowego w katalogu z którego kod został uruchomiony, powstaną dwa pliki o nazwach w postaci:

- <nazwa_skryptu>.o<id_zadania> - zawierający wyjście standardowe
- <nazwa_skryptu>.e<id_zadania> - zawierający wyjście błędu

Zadania MPI

Wybór implementacji MPI

Wyświetlenie listy dostępnych implementacji:

```
mpi-selector --list
```

Wybór implementacji (interaktywny):

```
mpi-selector-menu
```

Wybór implementacji (z linii poleceń):

```
mpi-selector --user --set=<pełna_nazwa_implementacji>
```

Kompilacja

Kompilację, podobnie jak w przypadku zadań OpenMP można wykonać bądź poprzez uzyskanie bezpośredniego dostępu do linii poleceń węzła obliczeniowego (zadanie interaktywne), bądź umieścić w skrypcie wysyłanym jako zadanie wsadowe.

Kompilacja przykładowego kodu, poleceniem:

```
labwn01$ mpicc –o mpi_bin hello_mpi.c
```

Natomiast jego uruchomienie:

```
labwn01$ mpirun --hostfile $PBS_NODEFILE -np <N> --mca btl
tcp,openib,self,sm --mca btl_openib_if_include mlx4_0:1 ./mpi_bin
```

Uruchamianie zadań MPI

W skrypcie wysyłanym do systemu Torque należy umieścić następujące polecenia:

Przejście do katalogu z którego wykonaliśmy zadanie (zbędne w przypadku podania ścieżek bezwzględnych dla polecenia **mpirun**).

```
cd $PBS_O_WORKDIR
```

W przypadku korzystania z Ethernetu do komunikacji międzyprocesowej, przykładowa linijka będzie wyglądała następująco (wersja dla 32 rdzeni):

```
mpirun --machinefile $PBS_NODEFILE -np 32 --mca btl tcp,self,sm
./mpi_bin
```

W przypadku chęci skorzystania z Infinibandu:

```
mpirun --machinefile $PBS_NODEFILE -np 32 --mca btl openib,self,sm -
-mca btl_openib_if_include mlx4_0:1 ./mpi_bin
```

Wersja alternatywna (pozwala na automatyczny wybór pomiędzy Ethernetem a Infinibandem):

```
mpirun --machinefile $PBS_NODEFILE -np 32 --mca btl
tcp,openib,self,sm --mca btl_openib_if_include mlx4_0:1 ./mpi_bin
```

Gdzie:

- **-np** – całkowita ilość rdzeni zaalokowana dla zadania (suma rdzeni ze wszystkich przydzielonych maszyn obliczeniowych);
- **--machinefile** – nazwa zmiennej środowiskowej zawierającej listę przydzielonych serwerów obliczeniowych (wartość powinna być ustawiona na $PBS_NODEFILE gdyż jest to zmienna automatycznie eksportowana przez system Torque);
- **./mpi_bin** – nazwa do skompilowanego kodu, który chcemy wykonać.