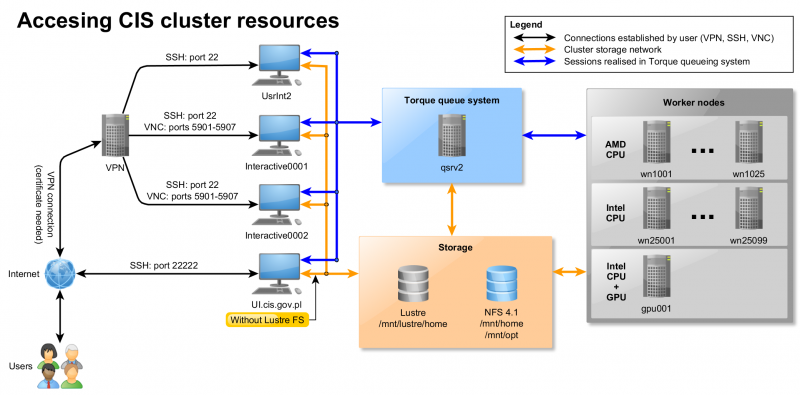
****

**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 |
| **qstatx** | Nakładka na polecenie *qstat* |
| **qnodesx** | Nakładka na polecenie *qnodes* |

**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. W przypadku braku tego argumentu zostanie wybrana kolejka domyślna.

Przykładowe wywołanie polecenia *qsub*:

qsub test.ah *#Skrypt o nazwie 'test.sh' uruchomiony w kolejce domyślnej/CPU domyślne/zasoby domyślne*

qsub -q a3d test.sh *#Skrypt o nazwie 'test.sh' uruchomiony w kolejce 3 dniowej/CPU: AMD/zasoby domyślne*

qsub -q a7d test.sh *#Skrypt o nazwie 'test.sh' uruchomiony w kolejce 7 dniowej/CPU: Intel/zasoby domyślne*

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 -q i12h -l nodes=1:ppn=4 test.sh *#Kolejka 12 godzin/CPU: Intel/1 węzeł obliczeniowy/4 rdzenie*

qsub -q a12h -l nodes=1:ppn=1 test.sh *#Kolejka 12 godzin/CPU: AMD/1 węzeł obliczeniowy/1 rdzeń*

qsub -q i14d -l nodes=1:ppn=40 test.sh *#Kolejka 14 dni/CPU: Intel/1 węzeł obliczeniowy/40 rdzenie - rezerwacja całego węzła*

qsub -q a14d -l nodes=1:ppn=64 test.sh *#Kolejka 14 dni/CPU: AMD/1 węzeł obliczeniowy/64 rdzenie - rezerwacja całego węzła*

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

qsub -q i3d -l nodes=1:ppn=64 test.sh *#Źle: Zbyt duża ilość rdzeni (procesory Intel posiadają ich max 40)*

qsub -q i2d -l nodes=1:ppn=1 test.sh *#Źle: Nieprawidłowa nazwa kolejki, kolejka 'i2d' nie istnieje*

qsub -q i7d -I *#Źle: Zadania interaktywne mogą być uruchamiane tylko w kolejkach 12 godzinnych*

qsub -q a7d -l nodes=1:ppn=100 test.sh *#Źle: Zbyt duża ilość rdzeni (procesory AMD posiadają ich max 64)*

qsub -q i12h -l nodes=40:ppn=1 test.sh *#Prawdopodobnie źle: zamieniona miejscami ilość węzłów/rdzeni*

W tym wypadku chcemy uruchomić skrypt test.sh do wybranej kolejki. 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. Po zakończeniu zadania pojawią się w naszym katalogu domowym dwa pliki:

test.sh.o27

test.sh.e27

Pierwszy zawiera standardowe wyjście, drugi standardowe wyjście błędów. 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 MPI**

Submituje się je podobnie do sekwencyjnych (też qsub-em), ale w trochę bardziej skomplikowany sposób - musimy wyspecyfikować nasze zapotrzebowanie na zasoby. Żeby wiedzieć co w tej kwestii jest możliwe, trzeba znać fizyczną strukturę zasobów klastra. W naszym klastrze są to trzy 24-rdzeniowe komputery. Zapotrzebowanie na zasoby zgłasza się przełącznikiem -l i podaje liczbę maszyn (nodes) oraz rdzeni na maszynę (ppn). Przykładowe wywołanie wygląda tak:

qsub -q q12h -l nodes=2:ppn=4 runmpi.sh

W tym wypadku zażądałem 2 maszyn po 4 rdzenie w każdej. Scheduler postara się dla nas zaalokować potrzebną liczbę rdzeni. Obecnie nasza konfiguracja maui zaalokuje zawsze żądaną liczbę RÓŻNYCH węzłów (taką ma ustawioną opcję), ale w ogólnym przypadku tak być nie musi, czyli jeśli ppn będzie małe to może nas upchnąć na jeden węzeł. Piszę o tym bo może w przyszłości włączymy takie pakowanie, na razie dla powtarzalności testów jest dokładna alokacja. Oczywiście jeśli scheduler nie ma tylu zasobów ilu zażądał użytkownik to zadanie się nie uruchomi (będzie sobie stało w kolejce i czekało na lepsze czasy).

I tutaj kolejna uwaga: aby móc korzystać z MPI należy załadować wszystkie potrzebne zmienne środowiskowe, na przykład dopisując do swojego .bashrc coś takiego:

module load openmpi

lub

module load mvapich2

**UWAGA**  
Nie ma możliwości załadowania OpenMPI na maszynach dostępowych. Moduł ten należy załadować  
z poziomu skryptu uruchamiającego zadanie lub będąc zalogowanym na węźle w trybie interaktywnym.

Teraz przechodzimy do zawartości skryptu runmpi.sh. W moim przypadku zawiera on tylko jedną linię i jest nią uruchomienie zadania MPI:

mpirun -machinefile $PBS\_NODEFILE -mca btl openib,sm,self mpimm

W pliku którego nazwa znajduje się w zmiennej $PBS\_NODEFILE scheduler zapisał dla nas nazwy zaalokowanych maszyn (każda występuje tam tyle razy ile rdzeni na niej dostaliśmy, dlatego nie ma potrzeby podawania parametru -np dla mpirun). Kolejny parametr (-mca btl self,tcp) jest parametrem transportowym i mówi bibliotece OpenMPI jakiej warstwy transportowej ma używać. W naszym przypadku (mamy tylko Ethernet), musimy używać TCP pomiędzy węzłami (tcp), natomiast w obrębie węzłów może to być pamięć dzielona (self). Tak naprawdę to parametr ten nie jest konieczny, bo OpenMPI samo się w końcu zorientuje że nie ma porządniejszego sprzętu komunikacyjnego (np Infinibandu), ale oszczędzi nam to ekranu błędów w pliku wynikowym. Na końcu podajemy nazwę pliku wykonywalnego (ja nazwałem swój mpimm). I tutaj zakładam że on już istnieje i nie musimy kompilować go przed wykonaniem. Zresztą kompilację w obrębie tego samego zadania odradzam jako marnowanie zasobów (wszystkie CPU czekają aż jeden rdzeń łaskawie skompiluje co trzeba). Testowe zadania należy kompilować na węzłach obliczeniowych albo przygotowując specjalny skrypt do kompilacji albo poprzez [zadania interaktywne](https://doc.cis.gov.pl/cis/index.php/Uruchamianie_zada%C5%84_na_klastrze#Zadania_interaktywne), na usrint nie są dostępne moduły MPI (openmpi czy mvapich2). Czyli na przykład:

mpicc mpi\_mm.c -o mpimm

Lista przydzielonych przez scheduler węzłów obliczeniowych dla naszego zadania znajduje się w pliku $PBS\_NODEFILE. Liczba rdzeni obliczeniowych która została przydzielona jest zdefiniowana w zmiennej $NCPUS.

W przypadku uruchamiania zadań na kilku węzłach obliczeniowych konieczne może być wyeksportowanie odpowiedniego środowiska, szczególnie jeśli modyfikowaliśmy je poprzez "module load". W prosty sposób możemy sprawdzić jakie zmienne środowiskowe zostały zmodyfikowane poprzez załadowane moduły wywołując komendę "module show <nazwa\_modułu>/<wersja>". Następnie zmienne te eksportujemy przy pomocy parametru "-x" komendy mpirun. Poniżej przykład dla skryptu w języku Python korzystającego z modułu python-mpi4py:

mpirun -machinefile $PBS\_NODEFILE -mca btl openib,sm,self -x PATH -x LD\_LIBRARY\_PATH -x PYTHONPATH mpitest.py

Wspomniany w opisie [zadań sekwencyjnych](https://doc.cis.gov.pl/cis/index.php/Uruchamianie_zada%C5%84_na_klastrze#Zadania_sekwencyjne) katalog tymczasowy $TMPDIR nie jest dzielony pomiędzy różnymi węzłami obliczeniowymi. Toteż dla zadań które wymagają dostępu do tych samych plików wynikowych, należy pliki te umieszczać na dysku dzielonym np: gdzieś w katalogu $HOME.

**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 -q q12h -I

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

qsub -q q12h -l nodes=1:ppn=4 -I

**Forwardowanie X-ów przy zadaniach interaktywnych**

Na klastrze możliwe jest uruchamianie programów wyposażonych w interfejs graficzny, takich jak np. MATLAB. Aby uruchomić taki program na dowolnym węźle klastra należy zalogować się na usrint (lub interactive0001) przez SSH z uruchomionym forwardowaniem X-ów przy użyciu polecenia:

ssh nazwa\_uzytkownika@usrint -Y

Następnie wchodzimy na węzeł w trybie interaktywnym:

qsub -v DISPLAY=$DISPLAY -l nodes=1:ppn=64 -IX

Przykładowo jeżeli chcemy uruchomić Matlaba wpisujemy:

module load matlab

matlab

Po uruchomieniu powinniśmy po krótkiej chwili zobaczyć splash screen, a następnie okno programu.

**Lista zależności**

Polecenie qsub umożliwia zdefiniowanie listy zależności między-zadaniowych. Zależności takie pozwalają na określenie np. kolejności wykonania się poszczególnych zadań użytkownika, określenie warunków jakie muszą zostać spełnione aby zadanie się wykonało, etc.

Składnia:

qsub -W depend=<lista\_zależności>

Najpopularniejsze parametry dla listy zależności:

* **afterany:jobid[:jobid...]**

**Opis:** Zadanie zostanie uruchomione dopiero po **zakończeniu** się wszystkich wylistowanych w parametrze ID zadań (niezależnie od wyników ww. zadań).

**Przykład użycia:**

qsub -W depend=afterany:234 run.sh *#Uruchom dopiero po zakończeniu się zadania o id=234*

* **afterok:jobid[:jobid...]**

**Opis:** Zadanie zostanie uruchomione dopiero po **poprawnym zakończeniu** się wszystkich wylistowanych w parametrze ID zadań (zadanie poprawnie zakończone to takie, którego główny skrypt zwrócił w kodzie wyjścia wartość równą 0). W przypadku gdy chociażby jedno z podanych *"jobid"* nie zakończy się z zerowym kodem wyjścia, uruchomienie zadania z podaną zależnością zostanie anulowane przez system kolejkowy.

**Przykład użycia:**

qsub -W depend=afterok:234:456 run.sh *#Uruchom po pomyślnym zakończeniu się zadań o id=234 oraz 456*

* **afternotok:jobid[:jobid...]**

**Opis:** Zadanie zostanie uruchomione tylko w przypadku gdy wszystkie z wylistowanych ID zadań zakończą się niepoprawnie (z kodem wyjścia różnym od 0). W przypadku gdy chociażby jedno z podanych *"jobid"* zakończy się z zerowym kodem wyjścia, uruchomienie zadania z podaną zależnością zostanie anulowane przez system kolejkowy.

**Przykład użycia:**

qsub -W depend=afternotok:12:13:14 run.sh *#Uruchom gdy wszystkie z podanych zadań (12,13,14) zakończą się z kodem wyjścia różnym od 0*

* **after:jobid[:jobid...]**

**Opis:** Zadanie będzie mogło zostać uruchomione po wystartowaniu wszystkich wylistowanych zadań.

**Przykład użycia:**

qsub -W depend=after:1:2:13 run.sh *#Uruchom dopiero gdy wystartują zadania od ID=1,2 oraz 13*

**Tutorial: Często popełniane błędy**

Lista najczęściej popełnianych błędów:

* **Niewłaściwa ilość deklarowanych rdzeni** - należy zwrócić uwagę aby deklarowana ilość rdzeni nie była większa niż dostępna ilość rdzeni na maszynie docelowej. W przypadku zadeklarowania chęci użycia większej ilości rdzeni niż dostępna zadanie nie uruchomi się nigdy;
* **Spacje w nazwach katalogów** - należy zwracać uwagę aby nie stosować znaków spacji w nazwach katalogów uczestniczących w wykonywaniu zadań na klastrze. W przeciwnym razie wykonanie zadania zazwyczaj kończy się błędem (np. użytkownik nie dostaje wyników końcowych, skrypty się nie uruchamiają w ogóle, etc.);

**Dodatek A - parametry polecenia "qsub"**

W przypadku polecenia "qsub" można wykorzystać następujące parametry:

**I - interactive**

usrint$ qsub -I

Uruchamia zadanie w sposób interaktywny.

**q - wybór kolejki**

usrint$ qsub -q [nazwa\_kolejki]

Przypisuje zadanie do określonej kolejki. Nazwy kolejek podane w tabeli w 1. części tego artykułu.

**l - definicja zasobów**

usrint$ qsub -l nodes=1:ppn=32

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

usrint$ qsub -l host=wn1013.cis.gov.pl

Pozwala "dostać się" na konkretną maszynę obliczeniową.

**v/V - eksport zmiennych środowiskowych**

usrint$ 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.

usrint$ qsub -V

Wymusza eksport wszystkich zmiennych środowiskowych na maszynę docelową.

**t - zadania macierzowe**

usrint$ 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**

usrint$ qsub -k o

usrint$ qsub -k e

usrint$ 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**

usrint$ 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".

**Dodatek B - 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>.qsrv2.cis.gov.pl

**Dodatek C - uwagi związane z rdzeniami i pamięcią na węzłach obliczeniowych**

Na serwerze obowiązują tzw. **cpusets**. Oznacza to, że każde zadanie dostaje od systemu Torque tyle rdzeni/pamięci ile zadeklaruje. Przykładowo wysyłając zadanie poleceniem "*qsub abcd.sh*" dostaniemy **1 rdzień** i **minimalny** przydział pamięci (gdyż takie są wartości domyślne). Potrzebując większej ilości należy zadeklarować potrzebną ilość na etapie wysyłania zadania. Sposób delkaracji ilości węzłów opisano powyżej. Natomiast potrzebną pamięć deklarujemy za pomocą wpisu:

qsub -I -l mem=15GB

**UWAGA**  
Dostępna pamięć RAM podzielona jest na tzw. "banki" pamięci. Węzły wn1001-wn1030 zawierające 256GB pamięci ram, posiadają po 8 banków pamięci (32GB każdy). Zadeklarowana ilość pamięci RAM zostanie zatem zaokrąglona w górę. Z racji na sprzętową architekturę systemu, oraz w celu racjonalnego gospodarowania zasobami system kolejkowy ogranicza dostępną pamięć ram w zależności od ilości użytych rdzeni na węźle.

Maksymalna ilość pamięci jaką możemy zatem dostać (nawet jeżeli zadeklarowane zostało więcej) ograniczona jest zatem w zależności od deklarowanej ilości wymaganych CPU.

Poniżej przedstawiono zależność **maksymalnego** rozmiaru pamięci jaki możemy dostać w zależności od ilości rdzeni:

|  |  |
| --- | --- |
| **Deklarowana ilość rdzeni** | **Maksymalna ilość pamięci RAM dostępna dla zadania [GB]** |
| 1-8 | 32 |
| 9-16 | 64 |
| 17-24 | 96 |
| 25-32 | 128 |
| 33-40 | 160 |
| 41-56 | 192 |
| 57-64 | 256 |

Przykład:

Wysyłając zadanie linijką:

qsub -I -l mem=120GB

Dostaniemy jedynie 32Gb pomimo deklarowanych 120Gb, gdyż domyślnie do zadania przypisany jest tylko jeden rdzeń.

Chcąc otrzymać przykładowe 120Gb pamięci na węźle, należy zwiększyć także ilość CPU's. Poprawne zatem będzie polecenie:

qsub -I -l nodes=1:ppn=25 -l mem=120GB

gdyż **25** to minimalna ilość rdzeni, która obsłuży wymaganą ilość pamięci RAM (każda większa ilość CPU będzie oczywiście również działała prawidlowo).

**Dodatek D - uwagi związane prędkością CPU na węzłach**

Domyślna prędkość CPU dla maszyn wn1001-wn1030 wynosi 2.3GHz na rdzeń.

Na węzłach aktywne są funkcje **Turbo Core** oraz **Cool'n'Quiet**.

Obie te funkcje dostosowują aktualną prędkość rdzeni w procesorze do zapotrzebowania na moc obliczeniową. Oznacza to, że niewykorzystany rdzeń ma prędkość niższą od nominalnej (w celu oszędności energii - funkcja Cool'n'Quiet), natomiast w przypadku wzrostu zapotrzebowania na moc obliczeniową (uruchomione procesy działające na danym rdzeniu), prędkość danego rdzenia jest zwiększana. Prędkości z jakimi pracują rdzenie mogą przewyższać wartość nominalną (funkcja Turbo Core), jednakże ogranicznikiem jest tu całkowita ilość mocy jaką pobiera procesor (i tym samym ilość ciepła jakie generuje podczas swojej pracy). W praktyce oznacza to, że w przypadku niewielkiej ilości używanych rdzeni ich prędkości będą wyższe niż w przypadku zwiększania ilości uruchomionych procesów na danej maszynie.

Najwyższe prędkości uzyskuje się korzystając z 1-4 rdzeni (każda maszyna posiada 4 fizyczne procesory, po 16 rdzeni każdy). Każdy dodatkowy proces natomiast, zwiększa ilość generowanego ciepła a tym samym zmusza system do zmniejszenia prędkości rdzeni.

Efektywna prędkość rdzeni jest ściśle uzależniona od TDP procesora. Nie ma zatem reguły jasno określającej jakiej prędkości możemy oczekiwać w przypadku wysyłania zadań na klaster. Oto przykładowe prędkości zmierzone podczas testów maszyn:

|  |  |  |
| --- | --- | --- |
| **Ilość obciążonych rdzeni** | **Uzyskana prędkość na używanym rdzeniu/rdzeniach (GHz)** | **Adnotacje** |
| 1-4 | 3.1-3.2 | Proste obliczenia *(niskie obciążenie ALU)* |
| 4 | ~3.1 | LINPACK *(wysokie obciążenie ALU)* |
| 8 | 3.0-3.1 | Proste obliczenia *(niskie obciążenie ALU)* |
| 8 | ~2.9 | LINPACK *(wysokie obciążenie ALU)* |
| 16 | 2.8-3.0 | Proste obliczenia *(niskie obciążenie ALU)* |
| 16 | ~2.7 | LINPACK *(wysokie obciążenie ALU)* |
| 32 | 2.6-2.9 | Proste obliczenia *(niskie obciążenie ALU)* |
| 32 | ~2.6 | LINPACK *(wysokie obciążenie ALU)* |
| 64 | ~2.55 | Proste obliczenia *(niskie obciążenie ALU)* |
| 64 | ~2.45 | LINPACK *(wysokie obciążenie ALU)* |