

Hermes Calculating Network System

Dokumentacja techniczna

Maciej Bendkowski

12 stycznia 2011

Spis treści

1	Wstęp	3
2	Cele projektu	4
3	Architektura systemu	4
4	Polityka nazewnictwa	5
5	Protokoły komunikacyjne	6
5.1	Kody statusów	6
5.2	Client-Master Protocol (CMP)	7
5.3	Master-Master Protocol (MMP)	9
5.4	Master-Slave Protocol (MSP)	10
6	Charakterystyka aplikacji klienta	12
7	Charakterystyka aplikacji serwera	12
8	Charakterystyka aplikacji serwera wykonawczego	14

1 Wstęp

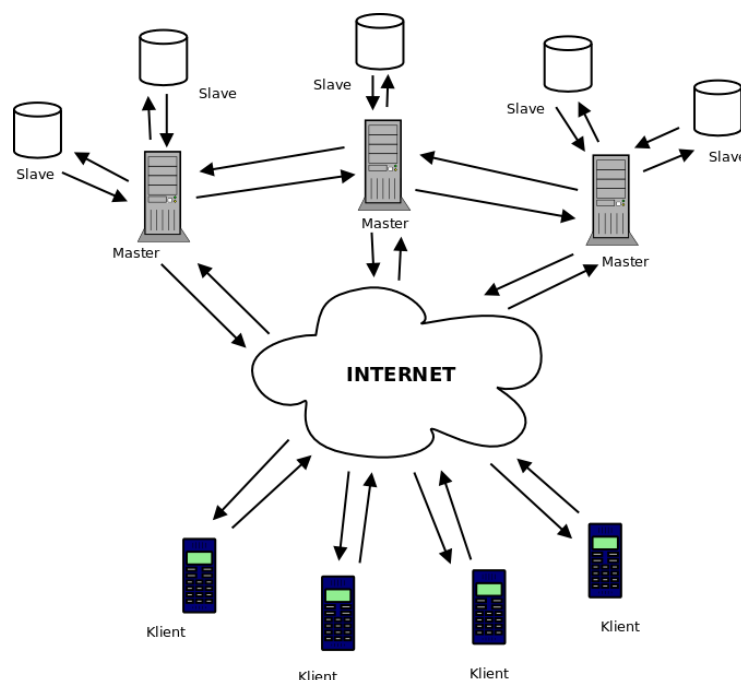
Hermes Calculating Network System (w skrócie Hermes-CNS, bądź Hermes) jest sieciowym systemem obliczeniowym opartym na Javie, stworzonym jako projekt zaliczeniowy w ramach przedmiotu Sieci Komputerowe prowadzonym przez dr Edwarda Szczypkę na Wydziale Matematyki i Informatyki Uniwersytetu Jagiellońskiego. Autorami projektu są Miłosz Lewandowski (Software Developer), Agnieszka Dymel (Software Developer), oraz Maciej Bendkowski (Project Manager, Programmer).

2 Cele projektu

Głównym celem projektu jest stworzenie sieciowego systemu obliczeniowego zdolnego do odbierania zleceń wymagających trudnych obliczeń, wykonywania ich wewnątrz systemu, oraz przekazywaniu z powrotem klientowi wyniku tychże obliczeń. Naszą grupą docelową są głównie klienci dysponującymi maszynami z ograniczonymi możliwościami wykonywania obliczeń, czy to ze względu na słabą moc procesora, czy to na niskie zasoby pamięci, czy to ze względu na brak odpowiednich operacji czy algorytmów potrzebnych do tych obliczeń. Ze względu na tak wybraną grupę docelową wygodnym modelem jest telefon komórkowy, czy inne małe urządzenie z dostępem do internetu. Poprzez Hermesa, chcemy skrócić czas oczekiwania klienta na wynik operacji w porównaniu z wykonaniem tych samych obliczeń na jego lokalnej maszynie, lub też, jeżeli lokalna maszyna nie jest w stanie ich wykonać, udostępnić je klientowi. Udostępniamy możliwość magazynowania wyniku operacji tak, że można wielokrotnie domagać się wyniku tej samej operacji bez konieczności ponownego wykonywania obliczeń. Dzięki takiemu rozwiązaniu zleceniodawca nie musi się pokrywać z odbiorcą wyników, przez co możliwe jest wysyłanie zlecenia z jednej maszyny, a odebranie wyniku na zupełnie innej maszynie. W ten sposób system udostępnia nam wynik naszego zlecenia kiedy tylko chcemy z praktycznie dowolnego miejsca na Ziemi.

3 Architektura systemu

Hermes-CNS jest podzielony na trzy części bazujących na architekturze Client-Master-Slave. Pierwszą częścią systemu jest aplikacja klienta (nazywana w skrócie Client). Jej głównym celem jest dostarczenie GUI dla klienta i ułatwienie mu tworzenie zleceń. Aplikacja klienta ma zapisywać odebrane wyniki i dostępne operacje. Drugą częścią systemu jest aplikacja serwera (w skrócie Master). Jej celem jest odbieranie zleceń, zarządzanie wykonywaniem, gromadzenie wyników zleceń, wysyłanie powiadomienia o dostępnych wynikach, oraz wysyłanie wyników do klienta. Ostatnią, trzecią częścią Hermesa jest aplikacja wykonawcza (w skrócie Slave). Jej celem jest odbieranie zleceń przekazanych przez Mastera, wykonanie ich lokalnie oraz odesłanie wyniku obliczeń do aplikacji serwera. Każdy Slave musi być połączony z jednym Masterem, który jednakże może posiadać wiele Slave'ów. Nie dopuszczamy sytuacji, kiedy jeden Slave posiada dwóch lub więcej Masterów. Aplikacje serwerowe tworzą pewien graf połączeń między sobą. Dopuszczamy możliwości wymiany informacji o sąsiadach pomiędzy aplikacjami serwerowymi, dzięki czemu umożliwiamy dynamiczne tworzenie i aktualizowanie grafu połączeń pomiędzy serwerami. Nie precyzujemy w jaki sposób aplikacje serwerowe są połączone fizycznie ze sobą, ani w jaki sposób aplikacje wykonawcze są połączone ze swoimi Masterami. Zakładamy natomiast na potrzeby projektu, że klienci łączą się z serwerami poprzez sieć internet, chociaż nie precyzujemy tej sytuacji w implementacji. Opisaną architekturę ukazuje poniższy schemat:



Rysunek 1: Architektura Hermes-CNS

4 Polityka nazewnictwa

W celu ujednolicenia i uściślenia wszelkich nazw własnych występujących wewnątrz Hermesa, wprowadzamy pewne reguły nazewnictwa, których będziemy się konsekwentnie stosować. Oto one:

1. Nazwy serwerów powinny nam je jednoznacznie rozróżniać. W tym celu na początku nazwy serwera umieszczamy kod kraju, w którym znajduje się serwer, np. PL dla Polski. Dalej umieszczamy nazwę miejscowości, w której znajduje się serwer, np. **Katowice**. Trzecią częścią nazwy jest słowo-klucz, definiujące nam kategorię operacji znajdujących się na serwerze. Jeżeli na serwerze znajdują się operacje z wielu różnych kategorii umieszczamy je po przecinku. Te nazwy nie muszą być bardzo precyzyjne, np. **Math** dla szeroko rozumianych operacji matematycznych jest wystarczająco satysfakcjonujące. Ostatnią częścią nazwy serwera jest pewien unikalny identyfikator serwera, np. imię i nazwisko administratora. Wszystkie części nazwy serwera oddzielamy znakiem “-”, który wolno stosować tylko w takim przypadku. Przykładowe poprawne nazwy serwerów:

- PL-Katowice-MATH-TCS@UJ
- DE-KOELN-DICTIONARY-nielubieponiedzialkow
- be-brussel-graph-AlexServer

2. Nazwy serwerów wykonawczych powinny w pierwszej części wskazywać na swojego Mastera. W dalszej części dopiero powinny określać typ operacji

dostępnych na Slave'ie. Wszystkie części nazwy serwera oddzielamy znakiem “-”, który wolno stosować tylko w takim przypadku. Przykładowe poprawne nazwy:

- TCS@UJ-Graph,Algorithms
- nielubieponiedzialkow-findFunctions

3. Nazwy operacji dostępnych na serwerze wykonawczym powinny jednoznacznie determinować co owe operacje liczą, np. **Boruvka's Algorithm**. Dodatkowo, każda operacja musi posiadać opis, w którym zawarte są: szczegółowa specyfikacja wejścia dla operacji, opis działania operacji (algorytmu), oraz specyfikacja wyjścia programu.
4. Nazwy numerów referencyjnych, używanych przy odbiorze zamówienia przez klienta, powinny być unikalne i jednoznaczne. Pierwsze 8 znaków numeru referencyjnego generujemy losowo tworząc unikatowy identyfikator. W drugiej części wpisujemy datę oraz czas wygenerowania numeru referencyjnego w formacie (rrrr/mm/dd/GG:MM:ss). W ostatniej części wpisujemy nazwę zamówionej operacji. Wszystkie części numeru referencyjnego oddzielamy znakiem “-”, który wolno stosować tylko w takim przypadku. Przykładowe poprawne numery referencyjne:

- aX8h99Aa-2010/10/29/12:34:21-Factor
- J9,fdsE2-2010/12/16/15:37:00-Boruvka's Algorithm

5 Protokoły komunikacyjne

Hermes-CNS korzysta z trzech różnych protokołów komunikacyjnych. Ze względu na swoje zastosowania, mniejsze wymagania sprzętowe, oraz łatwość w implementacji wszystkie protokoły komunikacyjne używane przez Hermesa są protokołami bezstanowymi. Omówimy teraz w szczególach każdy protokół z osobna, oraz opiszemy kody statusów.

5.1 Kody statusów

Kod	Opis
200	OK
300	Syntax error
301	Wrong IP address
302	No such operation
303	Wrong input data
304	Bussy server
305	Wrong reference number
306	Order forwarding

Tablica 1: Kody statusów

5.2 Client-Master Protocol (CMP)

Protokół ten używany jest w komunikacji pomiędzy klientem, a aplikacją serwera. Umożliwia wysyłanie zamówień do mastera, odbieranie przez klienta wyników zamówień, update'owanie tablicy dostępnych masterów, uzupełnianie tablicy operacji, jak i wypełnianie tablicy znanych masterów. Protokół udostępnia następujące komendy:

1. EXECUTE OPERATION:

```
EXECUTE OPERATION (/enter)
NAME: operation_name (/enter)
ON INPUT: (/enter)
input_data (/enter)
END INPUT (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje potwierdzenie przyjęcia zamówienia:

```
REFERENCE NUMBER (/enter)
NR: reference_nr (/enter)
ON SERVER: ip_address (/enter)
ONE TIME ORDER: TRUE/FALSE (/enter)
(/enter)
```

2. GET RESULT:

```
GET RESULT (/enter)
FOR: reference_nr (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje wynik zamówienia:

```
RESULT FOR REFERENCE (/enter)
NR: reference_nr (/enter)
PROCEEDING (/enter)
result_string (/enter)
END RESULT (/enter)
(/enter)
```

3. UPDATE OPERATION TABLE:

```
GET OPERATION TABLE (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje lista operacji:

```
FULL OPERATION TABLE (/enter)
PROCEEDING (/enter)
op_name (/enter)
DESCRIPTION (/enter)
op_description (/enter)
END DESCRIPTION (/enter)
(...)
END PROCEEDING (/enter)
(/enter)
```

4. UPDATE MASTER TABLE:

```
GET MASTER TABLE (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje lista serwerów:

```
FULL MASTER TABLE (/enter)
PROCEEDING (/enter)
master_name (/enter)
master_ip (/enter)
(...)
END PROCEEDING (/enter)
(/enter)
```

5. PING:

```
PING (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:


```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje potwierdzenie:

```
HERMES-CNS MASTER SERVER (/enter)
NAME: server_name (/enter)
PRIORITY: priority (/enter)
(/enter)
```

5.3 Master-Master Protocol (MMP)

Protokół ten używany jest w komunikacji pomiędzy serwerami. Umożliwia przekazywanie zamówień pomiędzy masterami, jak i dynamiczne mapowanie grafu połączeń masterów dążąc tym samym do sieci typu Full Mash. Protokół udostępnia następujące komendy:

1. FORWARD EXECUTION:

```
FORWARD EXECUTION (/enter)
FOR: recipient_ip (/enter)
DO NOT FORWARD TO (/enter)
master_ip (/enter)
(...)
END LISTING (/enter)

EXECUTE OPERATION
NAME: operation_name (/enter)
ON INPUT (/enter)
input_data (/enter)
END INPUT (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź, którą wysyłamy również do klienta:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje wynik zamówienia:

```
RESULT FOR REFERENCE (/enter)
NR: reference_nr (/enter)
PROCEEDING (/enter)
result_string (/enter)
END RESULT (/enter)
(/enter)
```

2. UPDATE MASTER TABLE:

```
GET MASTER TABLE (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje lista serwerów:

```
FULL MASTER TABLE (/enter)
PROCEEDING (/enter)
master_name (/enter)
master_ip (/enter)
(...)
END PROCEEDING (/enter)
(/enter)
```

3. PING:

```
PING (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

```
OK/ERROR status_code (/enter)
```

Jeżeli otrzymaliśmy odpowiedź OK, następuje potwierdzenie:

```
HERMES-CNS MASTER SERVER (/enter)
NAME: server_name (/enter)
PRIORITY: priority (/enter)
(/enter)
```

5.4 Master-Slave Protocol (MSP)

Protokół ten używany jest w komunikacji pomiędzy Masterem, a Slave'em. Umożliwia on przekazywanie zamówień do wykonania, update listy dostępnych operacji, oraz dynamiczne mapowanie dostępnych serwerów wykonawczych. Protokół udostępnia następujące komendy:

1. EXECUTE OPERATION:

```
EXECUTE OPERATION (/enter)
NAME: operation_name (/enter)
REFERENCE: reference_nr (/enter)
ON INPUT (/enter)
input_data (/enter)
END INPUT (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera wykonawczego, otrzymujemy odpowiedź:

OK/ERROR status_code (/enter)

Jeżeli otrzymaliśmy odpowiedź OK, następuje wynik zamówienia:

```
RESULT FOR REFERENCE (/enter)
NR: reference_nr (/enter)
PROCEEDING (/enter)
result_string (/enter)
END RESULT (/enter)
(/enter)
```

2. PING:

```
PING (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

OK/ERROR status_code (/enter)

Jeżeli otrzymaliśmy odpowiedź OK, następuje potwierdzenie:

```
HERMES-CNS SLAVE SERVER (/enter)
NAME: server_name (/enter)
(/enter)
```

3. UPDATE OPERATION TABLE:

```
GET OPERATION TABLE (/enter)
(/enter)
```

Po wysłaniu zapytania do serwera, otrzymujemy odpowiedź:

OK/ERROR status_code (/enter)

Jeżeli otrzymaliśmy odpowiedź OK, następuje lista operacji:

```
FULL OPERATION TABLE (/enter)
PROCEEDING (/enter)
op_name (/enter)
DESCRIPTION (/enter)
op_description (/enter)
END DESCRIPTION (/enter)
(...)
END PROCEEDING (/enter)
(/enter)
```

6 Charakterystyka aplikacji klienta

Z uwagi na przyjęty model maszyny w naszej implementacji aplikacji klienta, optymalizacja ruchu sieciowego, oraz zużycie pamięci będą naszym głównym priorytetem. Kierując się tymi optymalizacjami chcemy wprowadzić do aplikacji szereg funkcjonalności. Po pierwsze chcemy utworzyć dwa główne sposoby aktualizacji aplikacji, tj. tryb *manualny* oraz *automatyczny*. W trybie *automatycznym* nasza aplikacja będzie sama dbała o sprawdzanie aktualnych danych sieci, dostępnych operacjach, Masterach, itd. W trybie *manualnym* klient sam będzie musiał dbać o te parametry poprzez dokonywanie odpowiednich update'ów. Po drugie chcemy utworzyć trzy niezależne sposoby (poziomy) składania zamówień przez klienta:

- Poziom najniższy, bezpośrednio korzystając z protokołu *CMP* wewnątrz terminala aplikacji,
- Poziom średni, korzystając z nakładki graficznej, która pozwala wybrać interesującą nas operację, wyświetla szczegółową specyfikację tej operacji, oraz udostępnia formularz, gdzie można wpisać dane wejściowe (bądź też importować je bezpośrednio z dowolnego pliku).
- Poziom najwyższy, korzystając ze specjalnych pluginów. Każdy plugin posiada wbudowane operacje, które są dostępne zdalnie na specjalnych serwerach. Operacje wewnątrz pluginu są tej samej kategorii. Pluginy również stanowią graficzne nakładki na terminal aplikacji, ale różnią się od zwykłych nakładek tym, że mogą posiadać specyficzne formularze, ułatwiające tworzenie danych wejściowych, np. (mogą rysować graf online, który potem będzie przełożony na dane wejściowe pasujące do operacji), jak i graficznie przedstawiać wynik operacji.

Trzecim elementem aplikacji klienta jest biblioteka operacji, którą klient może manualnie edytować oraz przeglądać. Z biblioteką operacji związana jest wyszukiwarka operacji, które można wyszukiwać po nazwie, czy też słowach kluczowych występujących w ich specyfikacji. Wyszukiwarka ma na celu ułatwiać znajdowanie operacji w formularzu nakładki graficznej. Ułatwia również wyszukiwanie operacji wewnątrz samej biblioteki. Kolejnym elementem aplikacji jest tablica mapująca, czy znany Master jest dostępny, czy też nie. Tablica ta jest w pełni edytowalna przez klienta. Klient może ręcznie sprawdzać dostępność Masterów, korzystając z komendy *PING*. Ponadto, tablicę mapującą oraz bibliotekę operacji można update'ować ręcznie za pomocą komendy *UPDATE OPERATION TABLE* oraz *UPDATE MASTER TABLE*. Ostatnim istotnymi elementami Klienta są *skrzynka odbiorcza* oraz *skrzynka nadawcza*. *Skrzynka odbiorcza* przechowuje odebrane wyniki zamówień. *Skrzynka nadawcza* przechowuje historię zamówień klienta. Obie skrzynki są w pełni edytowalne przez klienta.

7 Charakterystyka aplikacji serwera

Aplikacja serwera ze względu na swoje zastosowania nie musi posiadać rozbudowanego środowiska graficznego. W związku z tym wszelkie komendy są podawane bezpośrednio wewnątrz terminala aplikacji. Dodatkowo, serwer posiada okienko logowe, gdzie na bieżąco wyświetla się wszelkie czynności serwera. Co

pięć minut logi zostają automatycznie zapisane na dysk twardy. Ze względu na swoje przeznaczenie serwer składa się z wielu współgrających części. Oto lista zadań serwera:

- Wczytywanie i przetwarzanie poleceń z terminala
- Odbieranie zamówień od klientów i wrzucanie ich na kolejkę priorytetową do przetworzenia, oraz przekazywanie wyników operacji klientowi gdy ten ich żąda
- Przetwarzanie zamówień z kolejki priorytetowej i wysyłanie ich do odpowiednich serwerów wykonawczych
- Kontrola aktualizacji sieci połączeń z innymi serwerami, jak i ze swoimi serwerami wykonawczymi
- Aktualizacja dostępnych operacji
- Odbieranie i forwardowanie zamówień od innych serwerów
- Odbieranie wyników od serwerów wykonawczych zapisując je do bazy danych PostgreSQL i powiadamianie o wynikach klienta

Każde z powyższych zadań będzie wykonywane w innym wątku, ze względu na potrzebę równoległego działania. Aby zminimalizować ilość danych przesyłanych wewnątrz sieci serwerów, aplikacja tablicuje niektóre dane o sieci. W trakcie działania serwera tablice te będą automatycznie aktualizowane. Są to: tablica operacji wraz z adresami Slave'ów, które są w stanie je wykonać, tablica serwerów wykonawczych, oraz tablica innych Masterów z którymi aplikacja jest w stanie się połączyć. Oprócz tych tablic, Master posiada połączenie z bazą danych PostgreSQL gdzie trzyma wyniki zamówień klientów. Ponadto, chcemy aby za pomocą poleceń terminala można było konfigurować serwer. Lista dostępnych komend:

1. `add slave/master (slave/master ip) (slave/master name)`
2. `remove slave/master (slave/master name)`
3. `clear slave/master/operation/result -table`
4. `update operation/master/slave -table`
5. `show operation/master/slave/result -table`
6. `wake slave/master (slave/master name)`
7. `sleep slave/master (slave/master name)`
8. `enable/disable autoupdate`
9. `enable/disable one-time-order`
10. `configure`
11. `shutdown/boot`
12. `save`
13. `help`

8 Charakterystyka aplikacji serwera wykonawczego

Aplikacja serwera wykonawczego jest najprostszym elementem Hermesa. Z tego też powodu cała aplikacja posiada tylko na dwa zadania:

- Odbieranie i kolejkowanie zamówień od Mastera
- Wykonywanie zamówień oraz odsyłanie ich wyniku z powrotem do Mastera

Aby umożliwić efektywne wykonywanie zamówień, Slave posiada tablicę dostępnych operacji wraz z ich fizycznym adresem na dysku. W celach diagnostycznych istotnym elementem aplikacji wykonawczej jest okienko logowe, gdzie zapisywana jest cała aktywność aplikacji. Dodatkowo, podobnie jak aplikacja serwera, Slave posiada terminal, za pomocą którego można w bezpośredni sposób wydawać polecenia serwerowi wykonawczemu (w tym konfigurować aplikację). Lista dostępnych komend:

1. `add operation (name) (dir-to-description) (dir-to-program)`
2. `remove operation (name)`
3. `show operation table`
4. `clear operation table`
5. `sleep operation (name)`
6. `wake operation (name)`
7. `show master`
8. `configure`
9. `shutdown/boot`
10. `save`
11. `help`