Computational Cluster

Katarzyna Węgiełek
Paweł Własiuk
Kamil Sienkiewicz
Marcin Wardziński

30 stycznia 2015

Spis treści

Rozdział 1

Wstęp

Tematem projektu jest stworzenie dokumentacji dla klastra obliczeniowego. Zadaniem projektowanego przez nas systemu będzie prowadzenie obliczeń rozproszonych. Klaster będzie umożliwiał rozwiązywanie skomplikowanych problemów, wykorzystujących algorytmy o dużej złożoności czasowej, w szczególności rzędu $o(2^n)$. Jego najważniejszą rolą będzie odpowiednie rozdzielenie zadań pomiędzy różne elementy systemu tak, aby jak najbardziej optymalnie wykorzystywać moc obliczeniową komputerów, z których się składa. Istotne jest również zminimalizowanie ryzyka utraty danych w przypadku awarii któregoś z komponentów.

1.1 Elementy klastra obliczeniowego

Task manager - dzieli zadanie na podproblemy i oblicza rozwiązanie końcowe na podstawie rozwiązań częściowych

Computational node - rozwiązuje pojedynczy podproblem i odsyła do jego rozwiązanie częściowe do communications server'a

Communications server - przesyła podproblemy i rozwiązania częściowe między task manager'ami a computational node'ami oraz wysyła rozwiązanie zadania do computational client'a

Computational client - wysyła problem do communications server'a i oczekuje na otrzymanie rozwiązania

Task solver - moduł używany przez computational node do rozwiązania podproblemu oraz przez task manager do podzielenia zadania i znalezienia ostatecznego rozwiązania na podstawie wyników częściowych

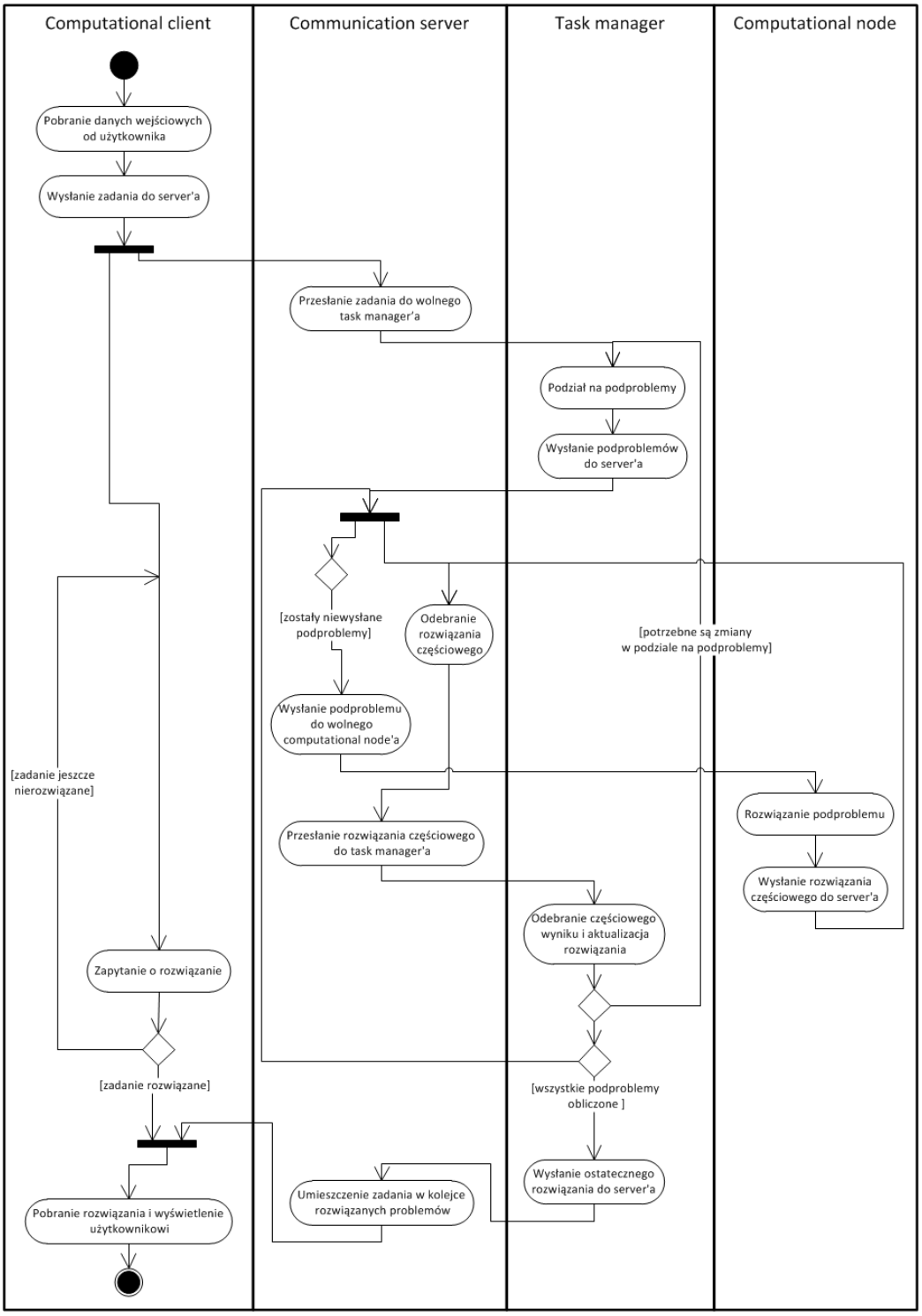
Rozdział 2

Opis działania systemu

Proces rozwiązywania zadania rozpoczyna się od wprowadzenia przez użytkownika danych wejściowych dla problemu. Robi to za pośrednictwem *computational client'a*. Jest to jedyny element klastra, z którym użytkownik ma bezpośredni kontakt. *Computational client* przetwarza wprowadzone informacje na komunikat i wysyła do *communications server'a* w formie pliku XML. Następnie zadanie trafia do kolejki zgłoszonych problemów. Jeśli istnieje w danej chwili w klastrze wolny *task manager*, który potrafi obsłużyć zadanie tego typu, to *communications server* wysyła do niego otrzymane dane wejściowe. *Task manager* dzieli problem na mniejsze części - podproblemy, przeznaczone do rozwiązania przez pojedyncze *computational node'y* i odsyła je do *server'a*. To czy *task manager* może podzielić dany problem zależy od tego, czy posiada *task solver*, zajmujący się poszukiwaną klasą problemów. *Communications server* wysyła odebrane podproblemy kolejno do wolnych *computational node'ów*, które są w stanie je rozwiązać (zawierają odpowiedni *task solver*). *Node'y* prowadzą obliczenia równolegle, a następnie wysyłają uzyskane wyniki do *server'a*. *Communications server* przesyła rozwiązania częściowe do *task managera*, który oblicza na ich podstawie ostateczny wynik. *Task manager* nie czeka aż wszystkie *node'y* zakończą obliczenia - zaczyna scalanie już po otrzymaniu pierwszych rozwiązań częściowych i aktualizuje rozwiązanie za każdym razem, gdy otrzyma kolejne dane z *server'a*. W niektórych typach zadań może się zdarzyć, że wyniki uzyskane przez *task manager'a* po scaleniu części rozwiązań spowodują zmiany w podziale na podproblemy - np. części z nich nie będzie się już opłacało rozwiązywać, bo wiadomo będzie, że nie wpłyną na ostateczny wynik. Gdy *task manager* uzy-

ska końcowe rozwiązanie, wysyła je do *server'a*. Trafia ono do kolejki zadań oczekujących na odebranie wyników. Kiedy *computational client* wyśle do *server'a* żądanie pobrania rozwiązania, *server* odsyła mu odpowiednie dane. Podczas całego procesu główny *communications server* synchronizuje trzymane przez siebie dane z *server'em backup'owym*, aby w razie awarii nie stracić części rozwiązań i zgłoszonych zadań. Jeśli *server* główny ulegnie uszkodzeniu i komunikacja z nim będzie niemożliwa, pozostałe komponenty nie przerwą swojej normalnej pracy i zaczną wymieniać komunikaty z *server'em backup'owym*.

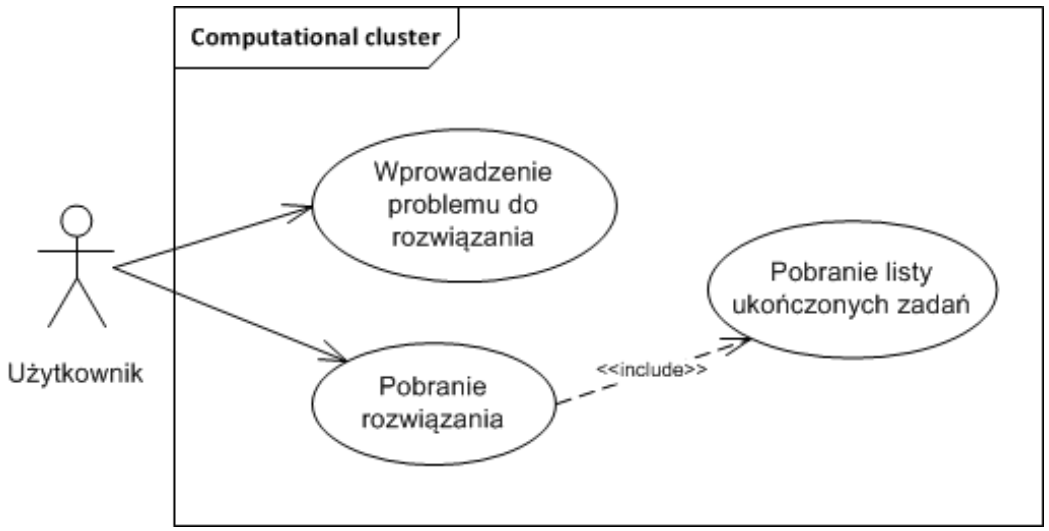
2.1 Diagram aktywności dla rozwiązania pojedynczego problemu



2.2 Diagramy przypadków użycia

2.2.1 Diagram przypadków użycia dla użytkownika

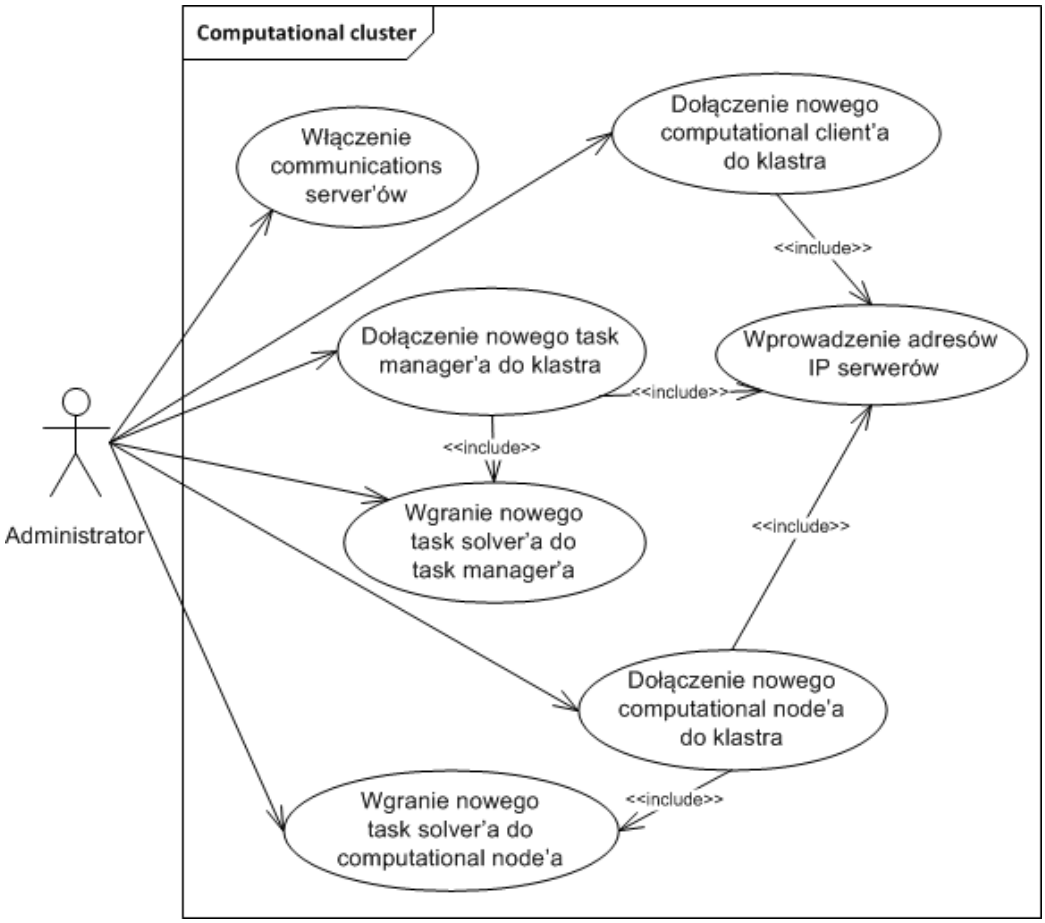
Główną rolą użytkownika jest zdefiniowanie problemu do rozwiązania przez klaster obliczeniowy. Wprowadza on dane wejściowe dla zadania za pośrednictwem *computational client'a*. Gdy obliczenia zostają zakończone, użytkownik pobiera z *communications server'a* rozwiązanie (również za pośrednictwem aplikacji klienckiej). Użytkownik w dowolnym momencie może sprawdzić, czy prace nad jego problemem zostały już zakończone. Na jego żądanie *communications server* wysyła do *computational client'a* listę wszystkich zadań, które zostały już rozwiązane. Jeśli interesujący go problem znajduje się na liście, użytkownik wskazuje identyfikator zadania i pobiera rozwiązanie.



2.2.2 Diagram przypadków użycia dla administratora

Administrator ma za zadanie skonfigurować wszystkie elementy klastra. Najpierw uruchamia *primery communications server* oraz *backup server*. Następnie dołącza do sieci pozostałe komponenty - *task manager'y*, *computational node'y* i *computational client'ów*. Aby umożliwić komunikację pomiędzy elementami systemu, administrator musi zapisać w pliku konfiguracyjnym przyłączanego elementu adresy IP obu serwerów. Żeby *task manager* i *computational node* mogły brać udział w rozwiązywaniu problemu, muszą mieć zainstalowany przynajmniej jeden *task solver*. Zatem przy dołączaniu ich do

klastra administrator powinien zainstalować odpowiednią wtyczkę.
Nowe komponenty mogą być dodawane zarówno przy tworzeniu klastra, jak i już podczas jego działania. Administrator może również w każdym momencie wgrać do *task manager'a* lub *computational node'a* kolejne *task solver'y*, odpowiadające nowym klasom problemów. Zwiększy się w ten sposób ilość typów zadań, jakie klaster jest w stanie rozwiązać.



Rozdział 3

Komunikacja

3.1 Nawiązywania połączenia

Uruchomienie całego systemu rozpoczyna się od uruchomienia *serwera komunikacyjnego*. Od tej chwili komponenty systemu tj. *węzły obliczeniowe* i *menadżery zadań* mogą zgłaszać swoją obecność w klastrze. Każdy z komponentów systemu posiada w swoim pliku konfiguracyjnym (w węzłach `MainServerAddress` i `BackupServerAddress`) **adres IP** i **port** *serwera komunikacyjnego* oraz serwera zapasowego. Zaraz po uruchomieniu każdy komponent zgłasza swoją obecność do głównego *serwera komunikacyjnego*. Po trzech nieudanych próbach nawiązania połączenia następuje wysłanie identycznej informacji do serwera backup'owego. Każdy z komponentów w takiej wiadomości informuje o swoim rodzaju podaje IP i port na którym działa. Na tej podstawie serwer będzie komunikował się z tymi elementami systemu.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ConnectionMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <ComponentType>TaskManager</ComponentType>
4   <IP>0.0.0.0</IP>
5   <Port>1111</Port>
6 </ConnectionMessage>
```

Listing 3.1: Wiadomość wysyłana przez komponent włączający się do systemu

Parametr `ComponentType` informuje o rodzaju komponentu systemu. Wartości, które może przyjąć ten parametr to:

- `TaskManager` - w przypadku gdy wiadomość pochodzi od *menadżera zadań*,
- `ComputationalNode` - gdy wiadomość pochodzi od *węzła obliczeniowego*

Zadaniem *serwera komunikacyjnego* jest utrzymywanie listy aktywnych komponentów systemu, oraz przechowywanie informacji na temat klas problemów, które dane komponenty obsługują. W tym celu serwer regularnie, co pewien określony czas będzie odpytywał wszystkie swoje komponenty prosząc o listę klas problemów możliwych do rozwiązania. W przypadku gdy takiej odpowiedzi nie otrzyma uznaje komponent za wyłączony i usuwa jego dane z pamięci. W przypadku otrzymania odpowiedzi na żądanie, *serwer* na podstawie otrzymanych danych uzupełnia/aktualizuje informacje o rozwiązywalnych problemach przez komponenty.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SolvableProblemListMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <ProblemClassList>
4     <ProblemClass problemClassName="name1" problemClassId="ee28fec2
          -6361-4a58-aaf1-b9ff0f509743"/>
5     <ProblemClass problemClassName="name2" problemClassId="f9ed0a8f-a9f1
          -494a-aea6-68ffc533934e"/>
6   </ProblemClassList>
7 </SolvableProblemListMessage>
```

Listing 3.2: Odpowiedź komponentu na żądanie serwera

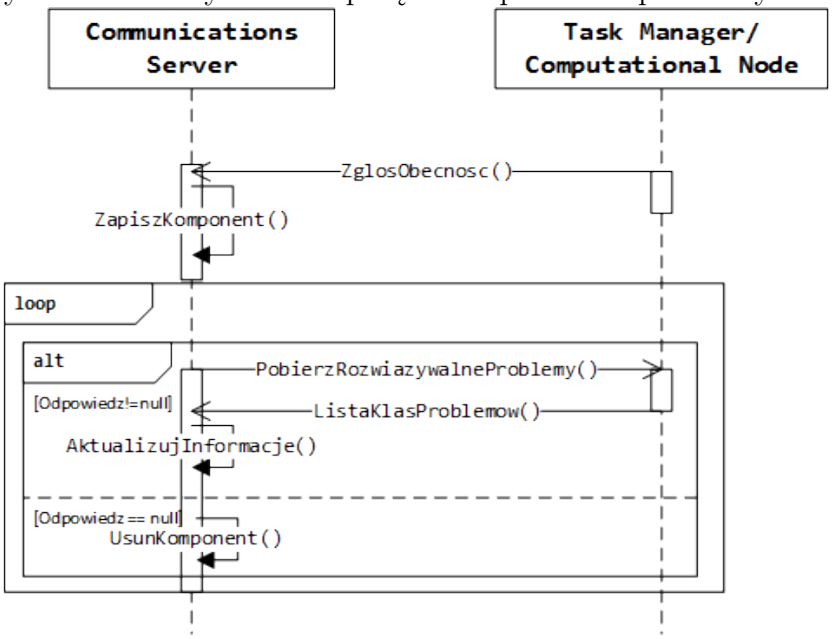
W wiadomości, dla każdego typu problemu zostają przekazane dwie informacje:

- `problemClassName` - nazwa problemu,
- `problemClassId` - guid problemu, który w jednoznaczny sposób identyfikuje typ problemu.

3.2 Lista rozwiązywalnych problemów

Proces rozwiązywania zadania przez *klaster obliczeniowy* rozpoczyna się na poziomie aplikacji klienckiej. Przed wysłaniem problemu do rozwiązania, apli-

Rysunek 3.1: Uzyskiwanie połączenia przez komponent systemu



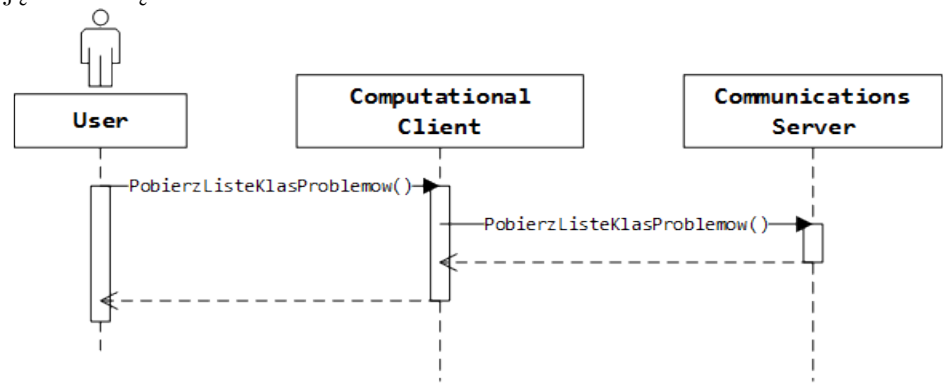
kacja kliencka musi pobrać z serwera informację na temat klas problemów rozwiązywalnych w danej chwili przez klaster obliczeniowy (wynika ona z obecnie dostępnych *Menadżerów zadań* i *węzłów obliczeniowych*). Aplikacja kliencka odpytuje *serwer komunikacyjny* o potrzebne informacje. *Serwer* w odpowiedzi na to żądanie odsyła wiadomość, w której zawarte są informacje o wszystkich typach problemów rozwiązywalnych przez klaster. Informacje zostają przetworzone i przedstawione użytkownikowi. Wiadomość przesłana do *aplikacji klienckiej* jest identyczna z tą, którą serwer otrzymuje od pozostałych komponentów systemu.

3.3 Rozwiązywanie zadania

Po wykonaniu czynności opisanych w poprzednim rozdziale, użytkownik może zlecić zadanie *klastrowi obliczeniowemu Aplikacja kliencka* wysyła do serwera wiadomość z informacją o typie rozwiązywanego problemu i dane wejściowe zadania.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

Rysunek 3.2: Diagram sekwencji pobierania informacji z serwera przez aplikację kliencką




```
<TaskOrderMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <problemClassId>4fbcbba1-014e-4643-b60f-f7888a95bb54</problemClassId>
  <Data>dane w postaci XML</Data>
</TaskOrderMessage>
```

Listing 3.3: Zlecenie rozwiązania zadania przez aplikację kliencką

Wiadomość zawiera 2 parametry konieczne do stworzenia i rozwiązania zadania:

- `problemClassId` - identyfikator klasy problemu, umożliwiający zidentyfikowanie, które części systemu są w stanie rozwiązać dany problem
- `Data` - parametry typu `String` zawierający dane wejściowe dla danego typu problemu. Dane te przedstawione są w formacie XML odpowiednim dla danego typu problemu.

Serwer po otrzymaniu wiadomości generuje specjalny identyfikator, który przypisuje do zadania. Będzie on wykorzystany do identyfikacji poszczególnych podzadań oraz umożliwi klientowi zidentyfikowanie zleconego zadania.

```
<?xml version="1.0" encoding="UTF-8"?>
<TaskTokenMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <TaskId>4fbcbba1-014e-4643-b60f-f7888a95bb54</TaskId>
</TaskTokenMessage>
```

Listing 3.4: Informacja z tokenem zwracana do aplikacji klienckiej

Taki token zostaje również odesłany do aplikacji klienckiej, która zapisze go w swoim pliku konfiguracyjnym. Poniżej przykład prezentujący odpowiedni wpis:

```
<TaskList>
  <Task taskname="customTaskName1" taskToken="ee28fec2-6361-4a58-
      aaf1-b9ff0f509743"/>
  <Task taskname="customTaskName2" taskToken="4fbcbba1-014e-4643-
      b60f-f7888a95bb54"/>
</TaskList>
```

Węzeł `Task` odpowiada jednemu zleconemu zadaniu przez użytkownika. Każdy taki węzeł posiada dwa atrybuty:

- `taskToken` - to identyfikator zadania nadany przez klaster obliczeniowy, dzięki niemu użytkownik będzie miał możliwość pobrania rozwiązania zadania,
- `taskname` - nazwa zadania nadana przez użytkownika, która umożliwi mu zidentyfikowanie zadania, nazwa obecna wyłącznie na poziomie aplikacji klienckiej.

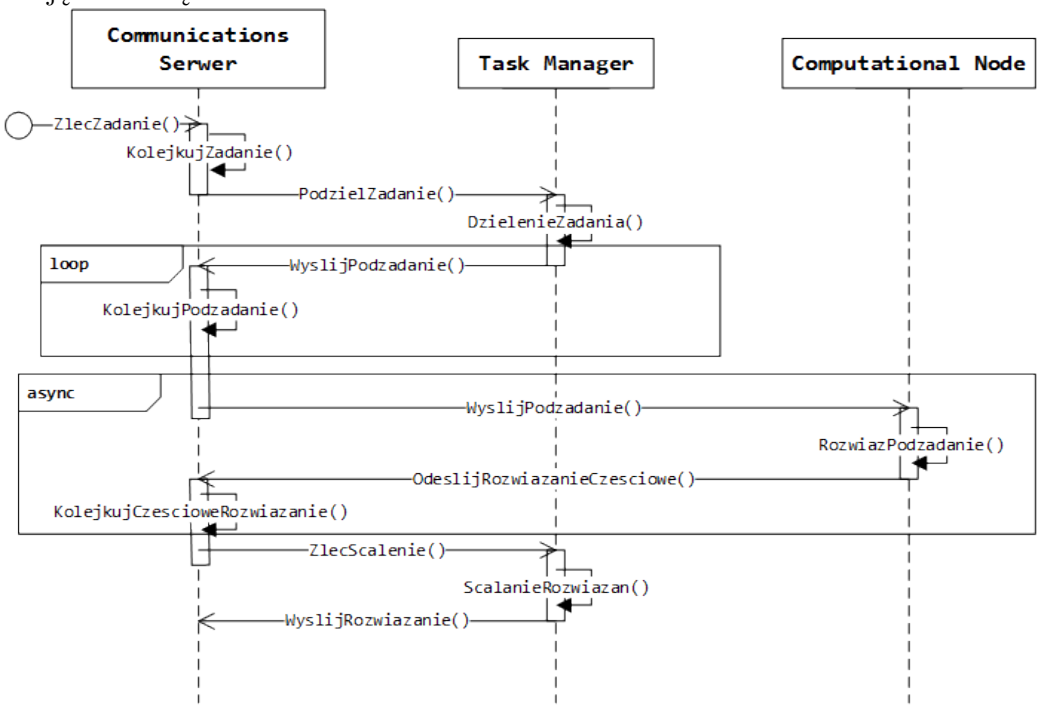
Klaster obliczeniowy wykonuje ciąg czynności koniecznych do otrzymania rozwiązania, tzn:

(1) przesłanie polecenia podziału zadania od *serwera* do *menadżera zadań* - wiadomość w formacie otrzymanym od *aplikacji klienckiej*,

(2) podział zadania i odesłanie stworzonych podproblemów prowadzących do otrzymania rozwiązania. *Menadżer zadań* dokonuje podziału zadania, każdemu z nich przydzielając unikalny identyfikator. Wiadomość zwrotna zawiera informacje o typie rozwiązywalnego problemu, identyfikatora zadania, identyfikatora podzadania, oraz danych wejściowych potrzebnych do rozwiązania zadania.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <SubtaskOrderMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <problemClassId>4fbcbba1-014e-4643-b60f-f7888a95bb54</
        problemClassId>
4   <TaskId>d104742a-76bb-4942-92bb-9f92149f4863</TaskId>
5   <SubtaskId>d104742a-76bb-4942-92bb-9f92149f4863</SubtaskId>
```

Rysunek 3.3: Diagram sekwencji pobierania informacji z serwera przez aplikację kliencką




```
6   <Data>dane podzadania XML</Data>
7 </TaskResultMessage>
```

Listing 3.5: Zlecenie podzadania

(3) przesłanie każdego z podzadań do *węzłów obliczeniowych*

(4) przeprowadzenie obliczeń i odesłanie częściowych rozwiązań do *serwera*. Odsyłana wiadomość zawiera informacje o typie rozwiązywalnego podproblemu, identyfikatorze zadania, identyfikatorze podzadania oraz rozwiązaniu częściowym problemu.

(5) przesłanie częsciowych rozwiązań do *menadżera zadań* w celu stworzenia rozwiązania. Przesłanie waidomości wcześniej zebranych od *węzłów obliczeniowych*.

(6) odesłanie pełnego rozwiązania zadania do *serwera* w identycznym formacie jak te, które serwer odsyła do *aplikacji klienckiej*

Zadanie przesłane do podziału przez *serwer komunikacyjny* jest w formacie otrzymanym od *aplikacji klienckiej*

3.4 Odczytanie rozwiązania

Proces odczytywania rozwiązania zadania rozpoczyna się od załadowania listy zadań zleconych przez daną aplikację z pliku konfiguracyjnego do którego wcześniej zostały wpisane identyfikatory poszczególnych zadań. Użytkownik wybiera zadanie którego rozwiązanie chce pobrać. Aplikacja odpytuje serwer o rozwiązanie, w wyniku czego otrzymuje informację na jego temat. Wiadomość, którą wysyła aplikacja kliencka w węźle `TaskId` zawiera identyfikator zadania którego rozwiązania oczekujemy.

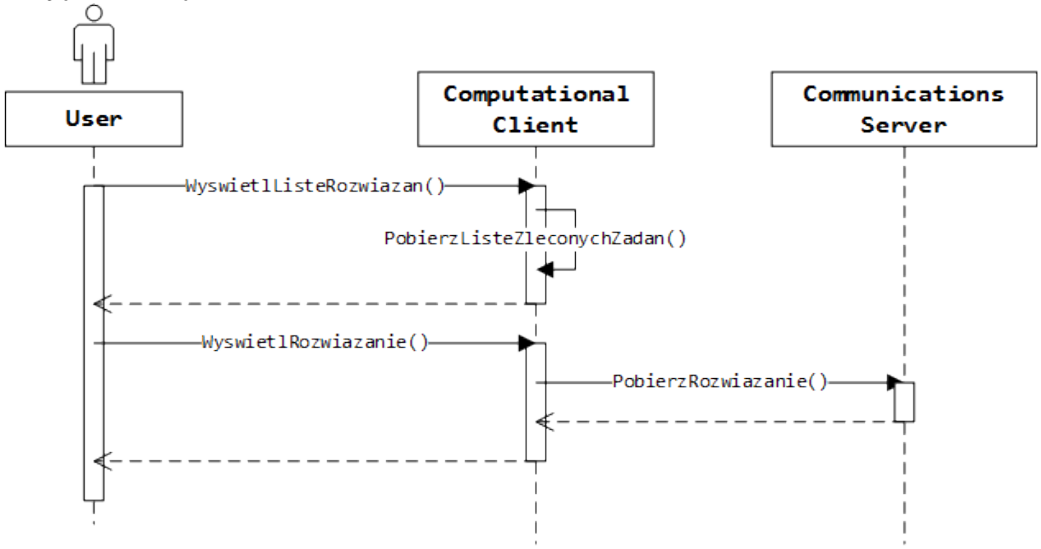
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <TaskResultMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <problemClassId>4fbcbba1-014e-4643-b60f-f7888a95bb54</problemClassId>
4   <TaskId>d104742a-76bb-4942-92bb-9f92149f4863</TaskId>
5   <Status>Done</Status>
6   <Data>rozwiazaniew postacie XML</Data>
7 </TaskResultMessage>
```

Listing 3.6: Rozwiązanie zadania

Wiadomość otrzymana od serwera składa się z 4 węzłów:

- `problemClassId` - identyfikator klasy problemów, której dotyczy zadanie. Informacja potrzebna do odpowiedniego wyboru pluginu, który odpowiednio sparsuje i przedstawi wyniki użytkownikowi.
- `TaskId` - identyfikator zadania
- `Status` - dwie możliwe wartości to `Done` - jeżeli zadanie zostało ukończone oraz `InProgress` - jeżeli rozwiązywanie zadania nadal trwa.
- `Data` - w przypadku gdy status przyjmuję wartość `Done` pole zawiera rozwiązanie zadania w postaci XML, w przeciwnym przypadku wartość jest pusta

Rysunek 3.4: Diagram sekwencji pobierania informacji z serwera przez aplikację kliencką



3.5 Schema

3.5.1 Uzyskiwanie połączenia przez komponenty

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="components">
    <xs:restriction base="xs:string">
      <xs:enumeration value="TaskManager"/>
      <xs:enumeration value="ComputationalNode"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ip">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{1,3}(\.[0-9]{1,3}){3}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="ConnectionMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="components" name="ComponentType"/>
```

```
      <xs:element type="ip" name="IP"/>
      <xs:element type="xs:integer" name="Port"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

3.5.2 Lista rozwiązywalnych problemów przez klaster obliczeniowy

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:simpleType name="guid">
  <xs:restriction base="xs:string">
   <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
       ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
       -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:element name="SolvableProblemListMessage">
  <xs:complexType>
   <xs:sequence >
    <xs:element name="ProblemClassList" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="ProblemClass" maxOccurs="unbounded"
           minOccurs="0">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute type="xs:string" name="
               problemClassName" use="required"/>
           <xs:attribute type="guid" name="problemClassId" use=
               "required"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
```

```
      </xs:sequence >
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.5.3 Zlecenie rozwiązania zadania

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
          -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="TaskOrderMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="guid" name="problemClassId"/>
        <xs:element type="xs:string" name="Data"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.5.4 Token wysyłany do aplikacji klienckiej

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
          -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="TaskTokenMessage">
    <xs:complexType>
      <xs:sequence >
```

```
12        <xs:element type="guid" name="TaskId"/>
13      </xs:sequence >
14    </xs:complexType>
15  </xs:element>
16 </xs:schema>
```

3.5.5 Żądanie o przesłanie rozwiązania zadania

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:simpleType name="guid">
4     <xs:restriction base="xs:string">
5       <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
            ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
            -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
6     </xs:restriction>
7   </xs:simpleType>
8
9   <xs:element name="TaskResultRequestMessage">
10    <xs:complexType>
11      <xs:sequence >
12        <xs:element type="guid" name="TaskId"/>
13      </xs:sequence >
14    </xs:complexType>
15  </xs:element>
16 </xs:schema>
```

3.5.6 Rozwiązanie Zadania

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:simpleType name="guid">
4     <xs:restriction base="xs:string">
5       <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
            ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
            -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
6     </xs:restriction>
7   </xs:simpleType>
8
9   <xs:simpleType name="status">
10    <xs:restriction base="xs:string">
11      <xs:enumeration value="Done"/>
```

```
      <xs:enumeration value="InProgress"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="TaskResultMessage">
    <xs:complexType>
      <xs:sequence >
        <xs:element type="guid" name="problemClassId"/>
        <xs:element type="guid" name="TaskId"/>
        <xs:element type="status" name="Status"/>
        <xs:element type="xs:string" name="Data"/>
      </xs:sequence >
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.5.7 Zlecenie rozwiązania podzadania

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
          -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="SubtaskOrderMessage">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="guid" name="problemClassId"/>
        <xs:element type="guid" name="TaskId"/>
        <xs:element type="guid" name="SubtaskId"/>
        <xs:element type="xs:string" name="Data"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3.5.8 Przesłanie rozwiązania podzadania

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
    qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="guid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
          ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a-fA
          -F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="SubtaskResultMessage">
    <xs:complexType>
      <xs:sequence >
        <xs:element type="guid" name="problemClassId"/>
        <xs:element type="guid" name="TaskId"/>
        <xs:element type="guid" name="SubtaskId"/>
        <xs:element type="xs:string" name="Data"/>
      </xs:sequence >
    </xs:complexType>
  </xs:element>
</xs:schema>
```