AKADEMIA GÓRNICZO-HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE



Wydział Inżynierii Metali i Informatyki Przemysłowej



PROJEKT ZALICZENIOWY

pt.

„Zagadnienie uzgadniania w systemach rozproszonych”

Zespół w składzie: **Krystian Cymerys**

**Konrad Boguń**

**Lucyna Hajder**

**Marcin Czanasz**

**Michał Góra**

**Karol Gałęzia**

Kierunek studiów: **Informatyka Stosowana**

Kraków 2016

**Spis treści**

[1. Analiza zagadnienia 4](#_Toc453134802)

[2. Algorytmy rozwiązujące zagadnienie uzgadniania 5](#_Toc453134803)

[2.1. Wybrane algorytmy 5](#_Toc453134804)

[2.2. Algorytm bizantyjskich generałów 6](#_Toc453134805)

[2.3. Algorytm randomizowany 6](#_Toc453134806)

[2.4. Algorytm jednorundowy 7](#_Toc453134807)

[2.5. Algorytm króla 7](#_Toc453134808)

[2.6. Algorytm królowej 8](#_Toc453134809)

[2.7. Algorytm dwóch armii 8](#_Toc453134810)

[2.8. k-set Consensus 9](#_Toc453134811)

[2.9. Terminating Reliable Broadcast 9](#_Toc453134812)

[2.10. Algorytm tratwy 10](#_Toc453134813)

[2.11. Rozgłoszeniowy algorytm konsensusu podstawowego 10](#_Toc453134814)

[2.12. Hierarchiczny algorytm konsensusu podstawowego 10](#_Toc453134815)

[2.13. Zestawienie najważniejszych informacji o wybranych algorytmach 11](#_Toc453134816)

[Rozgłoszeniowy algorytm konsensusu podstawowego 14](#_Toc453134817)

[Hierarchiczny algorytm konsensusu podstawowego 14](#_Toc453134818)

[3. Analiza porównawcza algorytmów 15](#_Toc453134819)

[3.1. złożoności obliczeniowej 15](#_Toc453134820)

[3.2. Złożoność czasowa: 16](#_Toc453134821)

[3.3. trudności implementacyjne 17](#_Toc453134822)

[3.4. znanych realizacji/wdrożeń 20](#_Toc453134823)

[4. Opis sposobu oceniania algorytmów oraz wybór najlepszego (?) 23](#_Toc453134824)

[5. Technologie rozproszone dostępne na rynku 24](#_Toc453134825)

[6. Analiza porównawcza implementacji 24](#_Toc453134826)

[7. Wnioski 24](#_Toc453134827)

[8. Bibliografia (oczywiście trzeba będzie ją poprawić) 25](#_Toc453134828)

# **Wstęp**

System rozproszony jest zbiorem samodzielnych komputerów, które są połączone za pośrednictwem sieci. Użytkownicy systemu rozproszonego mogą go odbierać jako jedno, zintegrowane środowisko obliczeniowe. Mogą one być użyte to realizacji interakcyjnych systemów komputerowych ogólnego przeznaczenia. Systemy rozproszone są w stanie zaoferować znaczące korzyści, do których można zaliczyć: dzielenie zasobów, otwartość, współbieżność, skalowalność, tolerowanie uszkodzeń oraz przezroczystość.

Współdzielenie zasobów wiąże się z tym, iż wielu użytkowników systemu może korzystać z danego zasobu. Przykładem takiego zasobu może być chociażby drukarka czy plik. Przezroczystość oznacza ukrycie przez użytkownikiem szczegółów rozproszenia np. lokalizacji lub implementacji zasobów. Transparentność ma zasadnicze znaczenie dla komfortu działania użytkownika oraz dla niezawodności oprogramowania. Przykładem przezroczystości jest Internet. Wysoka niezawodność wynika z możliwości użycia zasobów alternatywnych. W przypadku awarii jednego stanowiska w systemie rozproszonym pozostałe mogą kontynuować pracę. W przypadku kiedy system składa się z dużych, autonomicznych komputerów ogólnego przeznaczenia, to awaria jednego z nich nie wpływa na działanie pozostałych. W przeciwnym wypadku, czyli małe maszyny, odpowiedzialne za poszczególne funkcję, awaria może spowodować wstrzymanie działania całego systemu. Możliwość ciągłego rozwoju systemu (skalowalność) wynika z modularności systemu i otwartości sieci komunikacyjnej. Moc oraz możliwości przetwarzania wzrastają wraz z dodawaniem do systemu nowych zasobów. W praktyce skalowalność jest mocno ograniczona poprzez przepustowość sieci. Współbieżność również przynosi korzyści w postaci większej wydajności. W systemie rozproszonym można korzystać ze współbieżności między procesami klientów i między procesami serwerów.

# **Analiza zagadnienia**

Problem uzgadniania, czyli innymi słowy konkretny problem osiągnięcia konsensusu, polega na tym, iż każdy węzeł w systemie rozproszonym posiada swoją wartość początkową. Następnie wszystkie węzły muszą zgodzić się na jedną z wartości określonych poprzez poszczególne pozostałe węzły. W przypadku kiedy w systemie nie dochodzi do awarii algorytm rozwiązania problemu uzgadniania jest prosty do wytłumaczenia i zrozumienia, a mianowicie: każdy z istniejących węzłów w systemie wysyła wybraną przez siebie wartość, przypuśćmy decyzję, do pozostałych węzłów. Wykonywany jest wówczas algorytm wyboru większościowego. Rezultatem takiego działania jest osiągnięcie porozumienia pomiędzy węzłami, ponieważ dysponują one takimi samymi danymi oraz wykonują ten sam algorytm, co prowadzi do zdecydowania się na tą samą wartość vel decyzję.

W niniejszym problemie istnieje możliwość wystąpienia dwóch rodzajów awarii:

* Załamania (crash failures) – awaria ta polega na tym, iż pewien węzeł przestaje wysyłać komunikaty
* Awarie bizantyjskie – polegają na tym, że istnieje jeden lub więcej wadliwy węzeł, który wysyła dowolne komunikaty

Sformułowanie problemu uzgadniania można przedstawić w sposób obrazowy, jak to jest dokonywane w większości literatury. Z tego powodu na potrzeby niniejszego projektu również skorzystano z tego sformułowania:

*„Grupa armii bizantyjskich otacza miasta nieprzyjaciela. Rozkład sił jest taki, że jeśli wszystkie armie zaatakują razem, to będą w stanie zdobyć miasto. Innym sposobem uniknięcia porażki jest odwrót wszystkich armii. Generałowie poszczególnych armii mają zaufanych posłańców, którzy z powodzeniem dostarczą każdy komunikat od jednego generała do innego. Jednak niektórzy generałowie mogą być zdrajcami usiłującymi doprowadzić do porażki armii bizantyjskich. Należy opracować taki algorytm, który umożliwi wszystkim wiernym generałom uzgodnienie pewnego planu działania. Ostateczna decyzja powinna być z grubsza taka, jaka zostałby podjęta w drodze głosowania większościowego nad decyzjami poszczególnych generałów. W przypadku nierozstrzygnięcia głosowania końcową decyzją ma być odwrót.”* [[[1]](#endnote-2)].

W systemie rozproszonym odpowiednikiem generałów będą węzły, natomiast posłańcy będą odpowiadać kanałom komunikacyjnym.

//MOŻNA JESZCZE COŚ DOPISAĆ

# **Algorytmy rozwiązujące zagadnienie uzgadniania**

W niniejszym rozdziale omówione zostaną algorytmy rozwiązujące zagadnie uzgadniania. Przedstawiono tu krótką charakterystykę każdego z wybranych algorytmów. Rozdział ten prezentuje także podstawowe informacje związane z każdym z rozwiązań, takie jak dla zgodność, skończoność czy liczba kroków dla jakich jest poprawny.

## **Wybrane algorytmy**

Dokonując analizy problemu uzgadniania można stwierdzić, iż powstało wiele algorytmów rozwiązujących powyższy problemu. Oprócz dedykowanych algorytmów takich jak: algorytm generałów bizantyjskich, algorytm jednorundowy czy algorytm króla, istnieje szereg pozostałych algorytmów umożliwiających rozwiązanie problemu uzgadniania, sprowadzając go jednocześnie do problemu konsensusu. Wybrane algorytmy rozwiązujące zagadnienie uzgadniania zebrane zostały w poniższej tabeli wraz ze wskazaniem twórcy i daty powstania algorytmu.

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa algorytmu | Twórca | Rok powstania | Wybrany przez |
| Algorytm generałów bizantyjskich (Byzantine Generals Algorithm) [[[2]](#endnote-3)] | Leslie Lamport  Robert Shostak  Marshall Pease | 1982 | Krystian Cymerys |
| Algorytm króla (The Phase King Algorithm) [[[3]](#endnote-4)] | Piotr Berman  Juan Garay | 1989 | Lucyna Hajder |
| Algorytm jednorundowy |  |  | Marcin Czanasz |
| Algorytm randomizowany (randomized algorithm) [[[4]](#endnote-5)] | Michael Ben-Or | 1983 | Krystian Cymerys |
| Algorytm dwóch armii (Two Army/Generals Problem) [[[5]](#endnote-6)] | Eralp Abdurrahim Akkoyunlu  Kattamuri Ekanadham  R.V. Huber | 1975 | Lucyna Hajder |
| Algorytm królowej (The Queen Algorithm) [[[6]](#endnote-7)] | Piotr Berman  Juan Garay  Kenneth Perry | 1989 | Lucyna Hajder |
| Algorytm tratwy (Raft algorithm) [[[7]](#endnote-8)] | Diego Ongaro  John Ousterhout | 2014 | Karol Gałęzia |
| Rozgłoszeniowy algorytm konsensusu podstawowego |  |  | Michał Góra |
| Hierarchiczny algorytm konsensusu podstawowego[[8]](#endnote-9) | Akhil Kumar | 1990 | Michał Góra |
| k-set Consensus [[[9]](#endnote-10)] | Roberto De Prisco  Dahlia Malkhi  Michael L. Reiter | 1999 | Konrad Boguń |
| Terminating Reliable Broadcast[[[10]](#endnote-11)] | Sape J. Mullender | 1993 | Konrad Boguń |

**Tabela 1.** Wybrane algorytmy rozwiązujące zagadnienie uzgadniania. Źródło: [opracowanie własne]

## Algorytm bizantyjskich generałów

Algorytm bizantyjskich generałów jest dwurundowym algorytmem rozwiązywania problemu uzgadniania. Pierwsza runda przebiega analogicznie do algorytmu jednorundowego, a mianowicie każdy z generałów, w przypadku systemu rozproszonego węzłów, podejmuje decyzję, przesyła ją do wszystkich pozostałych węzłów oraz odbiera ich decyzje. Po zakończeniu tej rundy, w tablicy przechowującej decyzję „generałów” znajdują się wszystkie decyzje, które zostały podjęte przez każdy z węzłów. Druga runda niniejszego algorytmu opiera się o założenie, iż wcześniej wspomniane decyzje zostają przesłane do pozostałych generałów. Oczywistym jest, że generał nie musi przesyłać swoich decyzji do siebie samego, a także nie musi odsyłać innemu generałowi decyzji tego generała. Analizując powyższe założenia algorytmu, można zaobserwować, iż liczba komunikatów, które generał powinien przesłać zmniejsza o jeden wraz ze wzrostem rund. Procedura głosowanie odbywa się w dwóch etapach:

* Etap I – dokonanie wyboru na podstawie decyzji otrzymanej od danego generała oznaczonego jako G oraz przekazanych informacji przez pozostałych generałów dotyczących decyzji generała G. Wynik głosowania uznaje się za „prawdziwy” i zostaje zapisany do specjalnej tablicy, która przechowuje wyniki głosowania większościowego. Procedura jest powtarzana dla każdego generała z osobna.
* Etap II – ostateczna decyzja podejmowana jest na drodze wyboru większościowego spośród wyników znajdują się we wcześniej wspomnianej tablicy, przechowującej wyniki pierwszego głosowania.

Algorytm bizantyjskich generałów może ulegać zarówno załamaniom systemu jak również awariom bizantyjskim.

## Algorytm randomizowany

Algorytm randomizowany lub inaczej zwany randomizacyjnym działa przy założeniu, że większość procesów jest prawidłowych, czyli spełnia poniższy warunek: n>2f. Łatwo zauważyć, iż powyższe wymaganie jest konieczne dla każdego algorytmu, który rozwiązuje zagadnienie konsensusu w układach asynchronicznych, jak również rozwiązania problemu uzgadniania. Wymaganie to musi być spełnione, nawet wtedy gdy wszystkie procesy mają dostęp do generatora liczb losowych.

W algorytmie randomizowanym każda wiadomość opatrzona jest specjalnym znacznikiem zawierającym etykietę (R lub P), numer rundy oraz wartość, która przyjmuje 0 lub 1. Dodatkowo wartość dla etykiety P może przyjąć stan „?”. Wiadomości oznaczone etykietą R nazywane są raportami, natomiast oznaczone etykietą P – propozycjami. Każde wykonanie pętli nazywane jest rundą i każda z nich składa się z dwóch asynchronicznych faz. W pierwszej fazie, procesy przekazują wszystkich pozostałych procesom swoją wartość decyzji (0 lub 1). W drugiej fazie, jeżeli proces otrzymał od większości raporty z tą samą wartością, to proponuje tą wartość wszystkim pozostałym procesom. W przeciwnym wypadku proponuje wartość „?”. Na koniec drugiej fazy, jeżeli proces otrzymał f+1 propozycji z tą samą wartością różną od „?”, wtedy podejmuje decyzję o ustawieniu tej wartości. W przypadku kiedy otrzyma co najmniej jedną wartość różną od „?”, to przyjmuje ją jako nowy estymat, w przeciwnym wypadku przyjmuje losową wartość swoich oszacowań [[[11]](#endnote-12)].

## Algorytm jednorundowy

Każdy generał,podejmuje decyzję, przesyła ją do wszystkich pozostałych generałów oraz odbiera ich decyzje. Ostateczna decyzja jest podejmowana na drodze wyboru większościowego spośród wszystkich decyzji, zarówno własnej jak i otrzymanych od pozostałych. Załamanie kilku generałów może spowodować niemożność uzyskania porozumienia przez pozostałych, jeśli głosy dzielą się w miarę równo.

## Algorytm króla

Często jest mylony z algorytmem królowej. W porównaniu do algorytmu bizantyjskich generałów, ten algorytm wymaga znacznie mniejszej ilości przesłanych komunikatów do osiągnięcia porozumienia. Jednak jego wadą jest konieczność istnienia większej ilości wiernych generałów. Algorytm króla toleruje mniej niż n/3 błędów. Idea w tym przypadku jest taka sama jak idea algorytmu królowej. W każdej fazie jest inny król (wyznaczany metodą a priori). Jego decyzja jest ważniejsza niż innych. Faza posiada trzy rundy:

* Runda 1: podjęcie decyzji przez każdy proces i rozesłanie jej do wszystkich pozostałych.
* Runda 2: Jeżeli proces otrzyma wartość więcej niż n-f razy, to przesyła tą wartość dalej. Dodatkowo, jeśli takie propozycje pojawią cię więcej niż f razy, to proces zmienia wartość na tą w propozycji.
* Faza 3: Król przesyła swoją wartość do wszystkich. Jeżeli proces swoją wartość otrzyma mniej niż n-f razy, to zmienia ją na wartość otrzymaną od króla.

Takie fazy powtarzają się do osiągnięcia porozumienia. Pozytywną stroną algorytmu jest to, że niewielka ilość zdrajców nie jest w stanie wpłynąć na ostateczne glosowanie. Dzieje się tak, ponieważ ostateczna wartość jest proponowana przez poprawne procesy jedynie w przypadku, w którym tą wartość ma wiele procesów. Akceptacja wartości następuje, gdy większość procesów ma tą samą wartość.

## Algorytm królowej

Algorytm królowej jest mniej popularny od algorytmu króla. Jest to spowodowane potrzebą mniejszej ilości zdradliwych generałów. W tym sposobie możliwych jest występowanie jedynie n/4 wadliwych procesów.

W każdej fazie królową jest inny proces. Ideą jest takie wyznaczenie wartości, aby procesy wybrały tą samą wartość oraz żeby w przyszłych rundach proces nie zmieniał już swojej wartości. Każda faza dzieli się więc na dwie rundy

* Runda 1: każdy proces przesyła wszystkim swoją wartość. Następnie proces zmienia ją na taką, która wystąpiła najwięcej razy. Jeżeli zmieniona wartość wystąpiła ponad n/2 +f razy, to zapisuje ją. W przeciwnym razie nie wspiera jej.
* Królowa przesyła swoją wartość. Jeżeli proces nie wspiera żadnej wartości, to przyjmuje wartość otrzymaną od królowej.

Jeżeli podczas fazy królowa jest poprawna (generał wierny), to po jej zakończeniu wszystkie poprawne procesy mają wartość królowej. W takiej sytuacji w dalszych fazach żaden proces nie zmieni swojej wartości.

## Algorytm dwóch armii

Bardzo podstawowy algorytm, który odnosi się głównie do dwóch generałów. Pokazuje on trudność w osiąganiu porozumienia pomiędzy dwoma węzłami, a więc dotyczy zawodności kanałów komunikacyjnych, a nie uszkodzenia węzłów.

Problem ten można zobrazować jako dwie armie, które, rozstawione na wzgórzach otaczających dolinę, chcą zaatakować wroga w dolinie. Jeśli zaatakują jednocześnie, wygrają bitwę, w innym wypadku przegrają. W tym celu muszą się skomunikować ze sobą, jednak jedyna droga prowadzi przez dolinę – zawodny kanał z możliwością utraty wiadomości.

Jeżeli pierwszy generał wyśle wiadomość do drugiego, to nie ma pewności, że wiadomość ta została dostarczona. Aby potwierdzić otrzymanie wiadomości, drugi generał przesyła swoją odpowiedź do pierwszego dowódcy. Od tego momentu rozpoczyna się przesyłanie potwierdzeń otrzymania wiadomości przez obu dowódców. Żaden z nich jednak nie jest pewny, czy jego wiadomość dotarła. W ten sposób właśnie, nawet jeśli procesy są sprawne, osiągnięcie porozumienia pomiędzy nimi nie jest możliwe.

Algorytm dwóch armii polega więc na przesyłaniu komunikatów między sobą. Zawiera on komunikat ostateczny, który wyznacza koniec działania algorytmu. Wartością ostateczną jest wartość najczęściej występująca w komunikatach.

## k-set Consensus

K-set consensus uogólnia jednolity problem konsensusu (odpowiada przypadkowi k = 1). W tym problemie, każdy proces rozpoczyna swój udział w protokole z wartością wejściową i na końcu protokołu musi zdecydować się na jedną wartość tak, że co najwyżej k całkowite wartości są określane przez wszystkie poprawne procesy.

K set-consensus może być przydatny w przydzielaniu zasobów współdzielonych (np. przydział transmisji w sieciach komunikacyjnych). Taki protokół umożliwia procesom zgodzić się na małą liczbę częstotliwości do nadawania dużych ilości danych (np. film). Procesy mogą odbierać dane przy użyciu tej samej częstotliwości.

## Terminating Reliable Broadcast

Terminating Realible Broadcast to problem występujący w systemach rozproszonych, które obudowują nadawane wiadomości w zbiór otrzymanych procesów i ich błędów. W szczególności kiedy nadawca lub inny proces może spowodować awarię w dowolnym momencie.

Protokół TRB zarządza systemem w procesie wysyłania i zbiorze procesów otrzymanych, które mogą zawierać również nadawcę. Proces nazywany jest poprawnym jeśli wystąpi awaria w trakcie jego wykonywania. Celem protokołu jest przesyłanie wiadomości od nadawcy do zestawu procesów otrzymanych. Proces może wykonywać wiele operacji I/O podczas wykonywania protokołu, ale ostatecznie dostarcza wiadomość do aplikacji na tym procesie, który wywołał protokół TRB. Wszystkie poprawne odebrane procesy muszą dostarczyć wiadomość nadawcy jeśli nadawca również był poprawny. Proces odebrany może dostarczyć specjalny komunikat – „nadawca uszkodzony”, jeżeli nadawca uległ awarii. Prawidłowy proces jest więc gwarancją, że dane dostarczone do niego są również dostarczone do pozostałych poprawnych procesów.

Protokół TRB musi spełniać poniższe własności:

* Skończoność: każdy poprawny proces wykonuję dostarcza wiadomość
* Ważność: Jeśli nadawca wiadomości m nie jest uszkodzony to wszystkie poprawne procesy dostarczają wiadomość m
* Integralność: Każdy poprawny proces dostarcza co najwyżej jedną wiadomość. Jeśli proces dostarcza wiadomość różną od null, wtedy nadawca musi dostarczyć wiadomość m
* Zgodność: Jeśli poprawny proces dostarcza wiadomość m to każdy inny poprawny proces również dostarcza m.

Obecność błędów w systemie sprawia, że te właściwości są trudne do spełnienia

TRB jest ściśle powiązane z problemem konsensusu (nie jest identyczne).

## Algorytm tratwy

Algorytm tratwy (Raft algorithm) jest stosunkowo nowy i priorytetem w jego tworzeniu było, by był łatwy do zrozumienia. Jest równoważny algorytmowi Paxos w tolerancji błędów i wydajności.

Dostępne są 3 stany nodów: zwolennik (follower), kandydat (candidate) i lider (leader). Nody zaczynają jako zwolennicy. Jeśli zwolennicy nie dostają sygnałów od lidera, mogą stać się kandydatami. Kandydat żąda głosowania od pozostałych nodów. Nody odpowiadają swoimi głosami. Kandydat zostaje liderem jeśli dostanie większość głosów. Proces ten nazywa się Wyborami Lidera. Od teraz wszyskie zmiany przechodzą przez lidera. Każda zmiana jest dodawana jako wpis w logu noda. Wpis w logu jest zaakceptowany, więc nie uaktualni wartości noda. By zaakceptować wpis, node replikuje go do zwolenników. Następnie lider czeka, aż większość nodów napisze wpis. Wpis jest teraz zaakceptowany na nodzie lidera. Następnie lider informuje zwolenników, że wpis został zaakceptowany. Tym samym klaster doszedł do porozumienia co do stanu systemu. Proces ten nazywamy Replikacją Logów.

## Rozgłoszeniowy algorytm konsensusu podstawowego

Konsensusem nazywa się problem uzgadniania przez zbiór procesów jednej wartości spośród zbioru wartości zaproponowanych wstępnie przez te procesy. Rozgłoszeniowy algorytm konsensusu podstawowego zakłada dostępność mechanizmu podstawowego rozgłaszania niezawodnego i doskonałego detektora awarii. Koncepcja algorytmu przedstawia się następująco:

* Algorytm działa w rundach, zaś każda wiadomości jest oznaczona numerem rundy.
* Decyzja podejmowana jest, gdy proces otrzyma wiadomości od wszystkich poprawnych procesów i nie wykrył żadnej nowej awarii, a więc ma pewność, że otrzymał wszystkie propozycje, które będą wzięte pod uwagę przez wszystkie pozostałe procesy.
* Jeżeli wykryta została awaria, to proces nie ma pewności, czy posiadany przez niego zbiór wartości-propozycji jest taki sam, jak widziany przez wszystkie pozostałe procesy. Różnica może mieć na przykład miejsce, jeżeli jakiś błędny proces zdążył przesłać wiadomości tylko do części pozostałych procesów. W takim wypadku rozpoczynana jest kolejna runda.
* Do podjęcia decyzji używana jest dowolna deterministyczna funkcja, znana wszystkim procesom, np. wybierana może być najmniejsza wartość spośród zaproponowanych.

W algorytmie przesyłane są wiadomości dwóch typów. Typ MYPROP zawiera pole *proposedValue*, które jest zbiorem wartości zaproponowanych w danej rundzie przez poszczególne procesy, a także pole *roundNo*, określające numer rundy. Typ DECIDED natomiast, zawiera pole *decidedValue*, określające decyzję podjętą przez nadawcę wiadomości oraz pole *roundNo*, mówiące, w której rundzie podjęta została decyzja. Wiadomości typu DECIDED wysyłane są w momencie podjęcia decyzji przez proces (monitor). Znaczenie niektórych zmiennych jest następujące (pseudokod został przedstawiony w analizie algorytmów):

* *decidedi* zawiera wartość, na którą się zdecydował proces *Pi*, początkowo równa pewnej wyróżnionej wartości pustej.
* *proposalSeti* to zbiór, zawierający zestaw wszystkich widzianych przez proces (*monitor Qi*) propozycji.
* *roundNoi* to numer bieżącej rundy.
* *Correcti* to zbiór, początkowo obejmujący identyfikatory wszystkich procesów, oznaczający procesy, które według wiedzy dostępnej dla Pi są poprawne.
* *correctThisRoundi* zawiera identyfikatory procesów poprawnych w obecnej rundzie. Początkowo jest on pusty.

## Hierarchiczny algorytm konsensusu podstawowego

W Hierarchicznym algorytmie konsensusu podstawowego, każdy proces (monitor) pełni rolę lidera rundy. Przyjęto przy tym, że hierarchia procesów jest zdefiniowana przez uporządkowanie procesów według indeksów. Lider rundy rozgłasza swoją decyzję, która jest przejmowana przez pozostałe procesy, po czym zaczyna się kolejna runda z nowym liderem. Jeżeli lider ulegnie awarii, jego rolę przejmuje kolejny proces itd. Algorytm hierarchiczny konsensusu podstawowego (ang. hierarchical consensus) zakłada dostępność mechanizmu podstawowego rozgłaszania i doskonałego detektora awarii.

Jedynym używanym typem wiadomości jest typ DECIDED, niosący wartość podjętej decyzji w polu oraz numer rundy, w której podjęto decyzję, w polu . Algorytm używa następujących zmiennych (pseudokod został przedstawiony w analizie algorytmów):

* Wiadomość *decOuti* jest wiadomością, powiadamiającą inne procesy o podjętej decyzji przez lidera.
* *proposali* zawiera propozycję *Pi* procesu .
* *propRoundNoi* (ang. proposer round number) zawiera identyfikator procesu, który zaproponował bieżącą wartość zmiennej *proposali*.
* Tablica *deliveredi* zawiera wartość *True*, jeżeli proces (monitor) otrzymał powiadomienie o decyzji lidera.
* Tablica *lidersSeti* zainicjowana jest identyfikatorami wszystkich procesów, będących kolejnymi kandydatami na lidera. Identyfikator lidera -tej rundy jest zapisany w -tym elemencie tej tablicy. Tablica ta posiada identyczną zawartość dla wszystkich procesów, biorących udział w algorytmie.
* Tablica *broadcasti* zawiera wartość *True*, jeżeli monitor podjął już decyzję i rozgłosił ją wśród procesów.
* *roundNoi* zawiera numer kolejnej rundy.
* *suspectedi* reprezentuje zbiór procesów podejrzewanych o to, że są niepoprawne.
* Zmienna *k* jest używana lokalnie w obsłudze zdarzenia odbioru wiadomości DECIDED.

## Zestawienie najważniejszych informacji o wybranych algorytmach

|  |  |  |  |
| --- | --- | --- | --- |
|  | Algorytm generałów bizantyjskich (The Byzantine Generals Problem) [[[12]](#endnote-13)] | Algorytm randomizowany (Randomized algorithm) [[[13]](#endnote-14)] | Algorytm jednorundowy |
| skończoność (termination) | algorytm składa się z m+1 rund w pierwszej fazie, oraz z rundy podjęcia decyzji na podstawie otrzymanych danych w drugiej fazie. Rozwiązanie gwarantuje, że wszystkie poprawne procesy w końcu podejmą decyzję dochodząc tym samym do porozumienia | algorytm składa się z stałej liczby rund. Jeżeli niektóre procesy nie mają ustawionej losowo lokalnej wartości, to ustawiają ją taką samą jak pozostałe. | algorytm składa się z jednej rundy. |
| zgodność (agreement) | wartość decyzji wszystkich poprawnych procesów jest taka sama. Jeśli p­i oraz pjsą prawidłowe oraz podjęły decyzję, wówczas di = dj | zakładając, że niektóre poprawne procesy decydują o X. Proces ten musi uzyskać X z n-3f prawidłowych procesów. Każdy inny właściwy proces musi otrzymać X co najmniej n-4f razy (wszystkie poprawne procesy ustawiają swoje lokalne wartości X, oraz proponują ją, a także decydują się na X w następnej rundzie) | załamanie kilku generałów może spowodować niemożność uzyskania porozumienia przez pozostałych, jeśli głosy dzielą się w miarę równo. |
| integralność (integrity) | jeśli dowódca jest poprawny, to wszystkie poprawne procesy decydują się na wartość decyzji, którą zaproponował dowódca | jeżeli wszystkie poprawne procesy posiadają tą samą wartość X, otrzymają n‑2f propozycji zawierających X w pierwszej rundzie oraz będą decydować o X | - |
| dla jakich kroków jest niemożliwy | poprawny tylko dla ilości procesów n>3m (zdrajców powinno być 3 razy mniej) [[[14]](#endnote-15)] | poprawny tylko dla binarnych danych wejściowych. Nie działa w przypadku kiedy istnieją procesy bizantyjskie. Działa tylko wtedy gdy istnieją f<n/9 crash failures | - |
| ilość wiadomości | n\*[(n‑1)\*n‑k)\*(n‑k‑1)] wiadomości [] | (n-2f)(n-4f) | n(n-1) wiadomości.  Każdy generał wysyła i odbiera po jednej wiadomości do i od innych generałów. |
| złożoność obliczeniowa | O(Nf+1) – dla komunikatów niepodpisanych  O(N2) – dla komunikatów podpisanych [[[15]](#endnote-16)] | O(2n) – wszystkie poprawne procesy zdecydowały  O(1/2n-f) – najgorszy przypadek, kiedy wszystkie n-f poprawnych procesów wybiera tą samą wartość losowo | O(n) |

**Tabela 2.** Podstawowe informacje o wybranych algorytmach. Źródło: [11],[12],[13],[14].

|  |  |  |  |
| --- | --- | --- | --- |
|  | Algorytm króla (Phase King Algorithm)[[[16]](#endnote-17)] | Algorytm królowej (The Queen Algorithm)[[[17]](#endnote-18)] | Algorytm dwóch armii |
| skończoność (termination) | *Algorytm zawiera 3(f + 1)* rund dla (f+1) faz.  Jeżeli król jest prawidłowy, to wszystkie prawidłowe procesy mają tą samą wartość. | *Algorytm zawiera 2(f + 1)* rund dla (f+1) faz.  Jeżeli królowa jest prawidłowa, to wszystkie prawidłowe procesy mają tą samą wartość. | Rozwiązanie może być nieskończone. Każdy proces decyduje o wartości po skończonej ilości wiadomości.[[[18]](#endnote-19)] |
| zgodność (agreement) | Wszystkie procesy mają wpływ na ostateczną wartość (króla).  Jeśli wszystkie poprawne procesy mają wartość x na początku fazy , to  nadal będą mieć x w końcowym etapie. | Wszystkie procesy mają wpływ na ostateczną wartość (królowej).  Jeśli wszystkie poprawne procesy mają wartość x na początku fazy , to  nadal będą mieć x w końcowym etapie. | Wszystkie poprawne procesy mają decydować o tej samej wartości.  Wartość jest wybrana przez proces, jeśli wystąpiła n/2 razy[[[19]](#endnote-20)] |
| integralność (integrity) | Wszystkie poprawne procesy otrzymują >= n-f odpowiedzi.  Król ustawia swoją wartość na taką, która wystąpiła więcej niż n - 2f razy i przesyła ją do wszystkich procesów. | Wszystkie poprawne procesy otrzymują >= n-f odpowiedzi.  Królowa ustawia swoją wartość na taką, która wystąpiła więcej niż n/2 + f razy i przesyła ją do wszystkich procesów. | Każdy proces otrzymuje n-f odpowiedzi. Jeśli otrzymana wartość wystąpiła >n/2 ilość, to proces zmienia wartość i przesyła ją dalej. |
| dla jakich kroków jest niemożliwy | poprawny tylko dla ilości procesorów n>3f (zdrajców powinno być 3 razy mniej) | poprawny tylko dla ilości procesorów n>4f (zdrajców powinno być 4 razy mniej) | Poprawny tylko dla ilości procesorów n > 2f |
| ilość wiadomości | (f + 1)[(n − 1)(n + 1)] wiadomości[] | =(n-f)/2+(n-f)/2 > n/2+f wiadomości | Ilość wiadomości rośnie wykładniczo |
| złożoność obliczeniowa | O(n3) | O(n2) | O(n) |

**Tabela 3.** Podstawowe informacje o wybranych algorytmach cz.2. Źródło: [15],[16],[17],[18].

|  |  |  |
| --- | --- | --- |
|  | Terminating Reliable Broadcast [[[20]](#endnote-21)] | k-set Consensus [[[21]](#endnote-22)] |
| skończoność | Każdy poprawny proces dostarcza jakąś wiadomość | Każdy poprawny proces musi zdecydować się na jakąś wartość. |
| ważność | Jeśli nadawca wiadomości m nie jest uszkodzony to wszystkie poprawne procesy dostarczają wiadomość m | Jeśli jakiś proces decyduje się na jakąś wartość to musi być to wartość zaproponowana przez inny proces.  Jeden z warunków:  STRONG:  1. Decyzja dowolnego prawidłowego procesu jest równa wartości początkowej jakiego poprawnego procesu.  2. Jeśli wszystkie poprawne procesy zaczynają z wartością v to poprawne procesy decydują się na wartość v.  REGULAR:  1. Decyzja dowolnego prawidłowego procesu jest równa wartości początkowej jakiego procesu.  2.Jeśli wszystkie procesy zaczynają z wartością v to poprawne procesy decydują się na wartość v.  WEAK:  1.Jeśli nie wystąpiły błędy to decyzja dowolnego procesu jest równa wartości początkowej jakiegoś procesu.  2.Jeśli nie wystąpiły błędy i wszystkie procesy zaczęły z wartością v to każdy proces decyduje się na wartość v.  tl;dr Jeśli jakiś proces decyduje się na jakąś wartość to musi być to wartość zaproponowana przez inny proces. |
| integralność | Każdy poprawny proces dostarcza co najwyżej jedną wiadomość. Jeśli proces dostarcza wiadomość różną od null wtedy nadawca musi dostarczyć wiadomość m. |  |
| zgodność | Jeśli poprawny proces dostarcza wiadomość m to każdy inny poprawny proces rówież dostarcza m. | Wszystkie poprawne procesy muszą podjąć decyzję. Zbiór wartości na które procesy mogą się zdecydować wynosi co najwyżej k (>f) wartości. |
| złożoność obliczeniowa | D+ƒd | ? |
| dla jakich kroków jest poprawny | Poprawny proces zawsze otrzymuje wiadomość nawet jeżeli wystąpi błąd podczas wysyłania wiadomości (w tym przypadku proces dostaje wiadomość null). |  |

Tabela 4. Podstawowe informacje o wybranych algorytmach cz.3. Źródło: [19],[20].

|  |  |  |
| --- | --- | --- |
|  | Rozgłoszeniowy algorytm konsensusu podstawowego | Hierarchiczny algorytm konsensusu podstawowego |
| skończoność (termination) | Algorytm składa się z n+1 rund, a każda wiadomość jest oznaczona numerem rundy.  Wszystkie poprawne procesy ostatecznie decydują się na jakąś wartość. W systemie asynchronicznym, czas podejmowania decyzji jest przy tym skończony, ale nieznany.  Procesy wspólnie uzgadniają podejmowane decyzje, by ostatecznie zdecydowały się na jedną wartość. | Algorytm składa się z n+1 rund, a  każdy proces pełni rolę lidera rundy.  Procesy wspólnie uzgadniają podejmowane decyzje, by ostatecznie zdecydowały się na jedną wartość. |
| zgodność (agreement) | Wymaga się, by wszystkie procesy decydowały się na tę samą wartość.  Jeżeli żaden proces nie ulega awarii, procesy podejmują decyzję w pierwszej rundzie.  Nie istnieją dwa poprawne procesy, które zdecydowały się na inne wartości. Tym samym, poprawne procesy uzgadniają wspólną wartość.  W przypadku, w którym kolejno ulegają awarii wszystkie procesy, algorytm wymaga *n* rund. | Wymaga się, by wszystkie procesy decydowały się na tę samą wartość.  Lider rundy rozgłasza swoją decyzję, która jest przejmowana przez pozostałe procesy, po czym zaczyna się kolejna runda z nowym liderem.  Jeżeli lider ulegnie awarii, jego rolę przejmuje kolejny proces. |
| integralność (integrity) | Algorytm stosuje mechanizm podstawowego rozgłaszania niezawodnego oraz mechanizm doskonałego detektora awarii  Każdy proces decyduje się tylko raz. Procesy nie mogą zatem zmieniać raz podjętych decyzji.  Do podjęcia decyzji używana jest dowolna deterministyczna funkcja, znana wszystkim procesom. Na przykład wybierana może być najmniejsza wartość spośród zaproponowanych. | Algorytm stosuje mechanizm podstawowego rozgłaszania niezawodnego oraz mechanizm doskonałego detektora awarii  Hierarchia procesów jest zdefiniowana przez uporządkowanie procesów według indeksów. |
| dla jakich kroków jest niemożliwy | Algorytm przeznaczony jest dla modelu jawnych awarii. | Algorytm przeznaczony jest dla modelu jawnych awarii. |
| ilość wiadomości | Przypadek optymistyczny:  Przypadek pesymistyczny:  ) | Przypadek optymistyczny:  Przypadek pesymistyczny: |
| Złożoność obliczeniowa | Przypadek optymistyczny  Złożoność czasowa wynosi 1  Przypadek pesymistyczny  złożoność czasowa wynosi *n* | Przypadek optymistyczny  Złożoność czasowa wynosi 1  Przypadek pesymistyczny  złożoność czasowa wynosi *n* |

**Tabela 5.** Podstawowe informacje o wybranych algorytmach cz.4. Źródło: [dopisać]

# Analiza porównawcza algorytmów

## złożoności obliczeniowej

|  |  |  |
| --- | --- | --- |
| Algorytmy uzgadniania | Złożoność obliczeniowa | |
| Optymistyczna | Pesymistyczna |
| Algorytm bizantyjskich Generałów | O(Nf+1) | O(N2) |
| Randomized algorithm | O(2n) | O(1/2n-f) |
| Algorytm króla | O(n3) | ??? |
| Algorytm królowej | O(n2) | ??? |
| Algorytm dwóch armii | O(n) | ??? |
| Terminating Reliable Broadcast | D+ƒd | ??? |
| K-set Consensus | ??? | ??? |
| Algorytm jednorundowy | O(n) | ??? |
| Rozgłoszeniowy algorytm konsensusu podstawowego | O(1) | O(n) |
| Hierarchiczny algorytm konsensusu podstawowego | O(1) | O(n) |
| Algorytm tratwy (raft consensus algorithm) | ??? | ??? |

f - ilość rund komunikacji

N - ilość komunikatów

D -

## Złożoność czasowa:

|  |  |
| --- | --- |
| Algorytmy uzgadniania | Zakres stosowalności |
| Algorytm bizantyjskich Generałów | Wartość decyzji wszystkich poprawnych procesów jest taka sama. Jeśli p­i oraz pj są prawidłowe oraz podjęły decyzję, wówczas di = dj. Poprawny tylko dla ilości procesów n>3m (zdrajców powinno być 3 razy mniej) |
| Randomized algorithm | Procesy decydują o X, a jeden z nich musi uzyskać X z n-3f prawidłowych procesów. Każdy inny właściwy proces musi otrzymać X co najmniej n-4f razy, poprawny tylko dla binarnych danych wejściowych. Nie działa w przypadku kiedy istnieją procesy bizantyjskie. Działa tylko wtedy gdy istnieją f<n/9 crash failures. |
| Algorytm króla | Jeśli wszystkie poprawne procesy mają wartość X na początku fazy , to będą ją mieć w końcowym etapie. Poprawny tylko dla ilości procesorów n>3f (zdrajców powinno być 3 razy mniej) |
| Algorytm królowej | Jeśli wszystkie poprawne procesy mają wartość X na początku fazy, to będą ją mieć w końcowym etapie. Poprawny tylko dla ilości procesorów n>4f (zdrajców powinno być 4 razy mniej) |
| Algorytm dwóch armii | Poprawne procesy mają decydować o tej samej wartości. Wartość jest wybrana przez proces, jeśli wystąpiła n/2 razy. Poprawny tylko dla ilości procesorów n > 2f. |
| Terminating Reliable Broadcast | Jeśli nadawca wiadomości m nie jest uszkodzony to wszystkie poprawne procesy dostarczają wiadomość m. Poprawny proces zawsze otrzymuje wiadomość nawet jeżeli wystąpi błąd podczas wysyłania wiadomości (w tym przypadku proces dostaje wiadomość null). |
| K-set Consensus | Jeśli jakiś proces decyduje się na jakąś wartość to musi być to wartość zaproponowana przez inny proces. Wszystkie poprawne procesy muszą podjąć decyzję.  Zbiór wartości na które procesy mogą się zdecydować wynosi co najwyżej k (>f) wartości. |
| Algorytm jednorundowy | Załamanie kilku generałów może spowodować niemożność uzyskania porozumienia przez pozostałych, jeśli głosy dzielą się w miarę równo. n(n-1) wiadomości.  Każdy generał wysyła i odbiera po jednej wiadomości do i od innych generałów. |
| Rozgłoszeniowy algorytm konsensusu podstawowego | Jeżeli żaden proces nie ulega awarii, procesy podejmują decyzję w pierwszej rundzie.  W przypadku, w którym kolejno ulegają awarii wszystkie procesy, algorytm wymaga *n* rund. |
| [Hierarchiczny algorytm konsensusu podstawowego](http://wazniak.mimuw.edu.pl/index.php?title=Pr-1st-1.1-m13-Slajd25) | Lider rundy rozgłasza swoją decyzję, która jest przejmowana przez pozostałe procesy, po czym zaczyna się kolejna runda z nowym liderem. Jeżeli lider ulegnie awarii, jego rolę przejmuje kolejny proces. |
| Algorytm tratwy (raft consensus algorithm) | Lider wybierany jest głosem większości. Lider akceptuje wpisy w logu, rozsyła je na inne serwery i mówi im że mogą zaaplikować zmiany w swoich maszynach stanu. |

## trudności implementacyjne

|  |  |
| --- | --- |
| Algorytmy uzgadniania | Zakres stosowalności |
| Algorytm bizantyjskich Generałów | p1: decyzja[mójID] <- wybierzAtakLubOdrót  p2: dla każdego innego generała G  p3: send(G, mójID, decyzja[mójID])  p4: dla każdego innego generała G  p5: receive(G, decyzja[G])  p6: dla każdego innego generała G  p7: dla każdego innego generała G’ z wyjątkiem G  p8: send(G’, mójID, decyzja[mójID])  p9: dla każdego innego generała G  p10: dla każdego innego generała G’ z wyjątkiem G  p11: receive(G, G’, przekazywanaDecyzja[G, G’])  p12: dla każdego innego generała G  p14: wybórWiększościowy[mójID] <- decyzja[mójID]  p15: ostatecznaDecyzja <- większość(wybórWiększościowy) |
| Randomized algorithm | p1: x:=own input; r = 0  p2: Broadcast proposal(x, r)  p3: In each round r =1,2,…  p4: Wait for n-f proposals  p5: If at least n-­2f proposals have some value y  p6: x := y; decide on y  p7: else if at least n-­4f proposals have some value y  p8: x := y;  else  p9: choose x randomly with P[x=0] = P[x=1] = ½  p10: Broadcast proposal(x, r)  p11: If decided on a value -> stop |
| Algorytm króla | p1: In each phase i ϵ 1...f+1:  Round 1:  p2: Broadcast own value  Round 2:  p3: If some value x appears ≥n-f times  p4: Broadcast “Propose x”  p5: If some proposal received > f times  p6: Set own value to this proposal  Round 3:  p7: The king broadcasts its value  p8: If own value received < n-f proposals  p9: set own value to the king’s value |
| Algorytm królowej | p1: In each phase i ϵ 1...f+1:  Round 1:  p2: Broadcast own value  p3: Set own value to the value that was received most often  p4: If own value appears > n/2+f times  p5: support this value  else  p6: do not support any value  Round 2:  p7: The queen broadcasts its value  p8: If not supporting any value  p9: set own value to the queen’s value |
| Algorytm dwóch armii | p1: Initially Vi = {xi}  p2: for k = 1 to f+1 do  p3: send { v ∈ Vi : pi has not already send v} to all  p4: receive Vj from pj for all j (including i)  p5: Vi = U (n-1, j=0) Vj  p6: yi = min(Vi) |
| Terminating Reliable Broadcast | sender s in round 0:  p1: extract m  sender in round 1:  p2: send < m, sig( s ) > to all  Process p in round k, 1 <= k <= f+1  p3: if p extracted m from a valid message < m, sig(p1), … ,sig(pk-1) > in round k-1 and p≠sender then  p4: send <m,sig ( p1), ..., sig (pk-1), sig( p ) > to all  p5: receive round k messages from all processes  p6: for each valid round k message < m, sig ( p1), ..., sig(pk-1), sig (pk)> received by p  p7: if p has not previously extracted m then  p8: extract m  p9: if k = f + 1 then  p10: if in theentire execution p has extracted exactly one m then  p11: deliver( m )  p12: else deliver (SF)  p13: halt  http://www.cs.utexas.edu/~lorenzo/corsi/cs380d/past/03F/notes/10-30.pdf |
| K-set Consensus | p1: A process Pi, 1 < i < n, initiates k-set consensus:  p2: updatei (Obj[i]) with v;  p3: repeat  p4: local, array <- scan (Obj);  p5: until there are at least |N| - f non-null values in Obj;  p6: v <- max. of the values in local array  https://www.cs.uic.edu/~ajayk/Chapter14.pdf |
| Algorytm jednorundowy | p1: decyzja[mojID] <- wybierzAtakLubOdwrót  p2: dla każdego innego generała G  p3: send (G, mójID, decyzja[mójID])  p4: dla każdego innego generała G  p5: receive(D, decyzjap[G])  p6: ostatecznaDecyzja <- większość(decyzja) |
| Rozgłoszeniowy algorytm konsensusu podstawowego | p1: when e\_proposerRC( Pi, vi ) do  p2: proposalSeti = { vi }  p3: propOut.roundNO = roundNoi  p4: propOut.proposedValue = proposalSeti  p5: sendBRB (Qi, propOut)  p6: end when  p7: when e\_crash( P ) do  p8: correcti = correcti \ { Pj }  p9: end when  p10: when e\_receiveBRB (Qj, Qi, mS : MYPROP) do  p11: proposalSeti = proposalSetiᑌ mS.proposedValue  p12: correctThisRoundi = correctThisRound ᑌ { Pj }  p13: end when  p14: when (correct ᑕ correctThisRoundi) ᐱ (decidedi = nil) do  p15: if correctThisRoundi = correctLastRoundi then  p16: decidedi = MIN( proposalSeti )  p17: decideRC ( Pi, decidedi )  p18: decOut.roundNo = roundNoi  p19: decOut.decidedValue = decidedi  p20: sendBRB ( Qi, decOut )  p21: else  p22: roundNoi = roundNoi + 1  p23: correctLastRoiundi = correctThisRoundi  p24: correctThisRoundi = Ø  p25: propOut.proposedValue = proposalSeti  p26: propOut.roundNo = roundNoi  p27: send (Qi, propOut)  p28: end if  p29: end when  p30: when e\_receiveBRB (Qj, Qi, dIn: DECIDED) do  p31: if decidedi = nil then  p32: decidedi = dIn.decidedValue  p33: decideRC (Pi, decidedi)  p34: decOut.roundNo = roundNoi + 1  p35: decOut.decidedValue = decidedi  p36: sendBRB (Qi, decOut)  p37: fi  p38: end when  http://wazniak.mimuw.edu.pl/index.php?title=Pr-1st-1.1-m13-toc |
| [Hierarchiczny algorytm konsensusu podstawowego](http://wazniak.mimuw.edu.pl/index.php?title=Pr-1st-1.1-m13-Slajd25) | p1: when e\_proposerRC( Pi, vi ) do  p2: proposalSeti = { vi }  p3: end when  p4: when e\_crash( P ) do  p5: suscpectedi = suspectediᑌ { Pj }  p6: end when  p7: when liderSeti [RoundNoi] = Piᐱ proposali != nil ᐱ broadcast [roundNoi] = False do  p8: decideRC ( Pi, proposali )  p9: broadcast [roundNoi] = True  p10: decOut.decideValue = proposali  p11: decOut.roundNo = roundNoi  p12: sendBRB ( Qi, decOut )  p13: end when  p14: when liderSeti [roundNo] suscpectediᐯ delivered [roundNo] = True do  p15: roundNoi = roundNoi + 1  p16: end when  p17: when e\_receive (Qj, Qi, dIn : DECIDED) do  p18: if (i > dIn.roundNo > propRoundNoi) then  p19: proposali = dIn.decidedValue  p20: propRoundNoi = dIn.roundNo  p21: end if  p22: k = dIn.roundNo  p21: deliveredi[k] = True  p22: end when |
| Algorytm tratwy (raft consensus algorithm) | ᑕ ᑌ ᐱ Ø ᐯ ∈ |

## znanych realizacji/wdrożeń

|  |  |
| --- | --- |
| Algorytmy uzgadniania | Znane realizacje wdrożenia |
| Algorytm bizantyjskich Generałów  Algorytm Króla  Algorytm Królowej  Algorytm Dwóch Armii  Algorytm jednorundowy | * I[nterne](https://pl.wikipedia.org/wiki/Internet)t – błędy przekazywania pakietów w sieci wymusiły powstanie protokołu[TCP](https://pl.wikipedia.org/wiki/TCP_%28protok%C3%B3%C5%82%29), który zapewnia uzgodnienie wiedzy drugiego rzędu * K[ryptografi](https://pl.wikipedia.org/wiki/Kryptologia)a – zagadnienia zbliżone do[autoryzacji](https://pl.wikipedia.org/wiki/Autoryzacja_%28informatyka%29) stron * E[lektroni](https://pl.wikipedia.org/wiki/Elektronika)ka – zagadnienia przekazywania danych w warunkach wysokich zakłóceń, uzgadnianie wspólnych parametrów transmisji * S[ystemy wieloagentow](https://pl.wikipedia.org/wiki/System_wieloagentowy)e – uzgadnianie wiedzy agentów * T[eoria gier](https://pl.wikipedia.org/wiki/Teoria_gier) – zagadnienia współpracy między kilkoma graczami |
| Randomized algorithm | * W sieci reakcji chemicznych (ang. [chemical reaction network](https://en.wikipedia.org/wiki/Chemical_reaction_network)) * Objętość wypukłego ciała może być wyestymowana przez losowy algorytm z określoną precyzją w czasie wielomianowym * W złożoności komunikacji, równość dwóch stringów może zostać sprawdzona używając bitów komunikacji wraz z losowym protokołem |
| Przerywanie niezawodnego rozgłaszania  Terminating Reliable Broadcast | Brak **znanych** realizacji/wdrożeń.  https://www1.cs.fau.de/filepool/teaching/dependablesystems-2007/PDS\_20070510.pdf |
| K-set Consensus  Rozgłoszeniowy algorytm konsensusu podstawowego  [Hierarchiczny algorytm konsensusu podstawowego](http://wazniak.mimuw.edu.pl/index.php?title=Pr-1st-1.1-m13-Slajd25) | * Synchronizacja zegara (ang. [clock synchronization](https://en.wikipedia.org/wiki/Clock_synchronization)) * PageRank * Kontrola UAV (ang. [control of UAVs](https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle)) * Równoważenie obciążeń (ang. [load balancing](https://en.wikipedia.org/wiki/Load_balancing_%28computing%29) ) |
| Algorytm tratwy (raft consensus algorithm) | Systemy które implementują algorytm tratwy:   * Chubby * ZooKeeper * Spanner   Chubby i Spanner nie zostały opublikowane w szczegółach, choć twórcy obu twierdzą, że bazują na Paxos. Algorytm ZooKeeper został opublikowany w szczegółach, ale jest różny od Paxos.  -VIewstamped Replication Oki i Liskova - alternatywne podejście do problemu uzgadniania, stworzone mniej więcej wtedy co Paxos. VR korzysta z podejścia opartego na liderach, z wieloma podobieństwami do Raft.  Źródła:  http://ramcloud.stanford.edu/raft.pdf  -Systems that implement consensus algorithms, such as Chubby [2, 4], ZooKeeper [11, 12], and Spanner [6]. The algorithms for Chubby and Spanner have not been published in detail, though both claim to be based on Paxos. ZooKeeper’s algorithm has been published in more detail, but it is quite different from Paxos.  -Oki and Liskov’s Viewstamped Replication (VR), an alternative approach to consensus developed around the same time as Paxos. The original description [29] was intertwined with a protocol for distributed transactions, but the core consensus protocol has been separated in a recent update [22]. VR uses a leaderbased approach with many similarities to Raft. |

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa algorytmu | popularność (ilość wyników w google | pseudokod (ilość linijek) | znane realizacje/wdrożenia |
| algorytm bizantyjskich generałów | 265 000 | 15 | 5 |
| randomized algorithm | 176000 | 11 | 3 |
| algorytm dwóch armii | 237 000 | 6 | 5 |
| algorytm króla | 2630 | 9 | 5 |
| algorytm królowej | 4200 | 9 | 5 |
| terminating reliable broadcast | 981 | 13 | 0 |
| k-set consensus | 2470 | 6 | 4 |
| algorytm jednorundowy | 2 | 6 | 5 |
| rozgłoszeniowy algorytm konsensusu podstawowego | 90 | 38 | 4 |
| [Hierarchiczny algorytm konsensusu podstawowego](http://wazniak.mimuw.edu.pl/index.php?title=Pr-1st-1.1-m13-Slajd25) | 3 | 22 | 4 |
| raft algorithm | 2070 |  | 3 |

# Opis sposobu oceniania algorytmów oraz wybór najlepszego (?)

# Technologie rozproszone dostępne na rynku

W poniższym rozdziale przedstawione zostaną aktualne technologie rozproszone, które są dostępne na rynku informatycznym. Wyniki badania rynkowego zostaną przedstawione w formie tabeli z podstawowymi informacjami.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Socket | RPC | CORBA | DCOM | RMI | SOAP | .NET Remoting |
| Producent |  | Sun Microsystems | OMG | Microsoft | Sun Microsystems | wielu | Microsoft |
| Rok powstania |  | 1984 | 1991 | 1996 | 1997 | 2000 | 2002 |
| Obsługiwane języki | C, C++, Java, C#, Visual Basic, Python, Fortran | C | C, C++, Java, COBOL, Ada, Lisp, Python | C++. Visual Basic | Java, C# | C++, Java | Wszystkie obsługiwne przez .NET Framework |
| Obsługiwane platformy | Windows, Unix | Unix | Windows, Unix | Windows | Windows, Unix | Windows, Unix | Windows, Unix |
| Język opisu interfejsu |  | XDR | CORBA IDL | Microsoft IDL | Java | Zależnie od impl. | Zależnie od impl. |
| Protokół transportowy | TCP//IP  UDP/IP | TCP/IP, UDP/IP | IIOP | TCP/IP | JRMP | HTTP | TCP/IP, HTTP |
| Format komunikatów |  | binarny | binarny | binarny | binarny | tekstowy | binarny, tekstowy |
| Blokowany przez zapory sieciowe | tak | tak | tak | tak | tak | tak | nie (gdy korzysta z protokołu SOAP) |
| Standard kodowania |  | XDR | CDR | NDR | serializacja Java | XML | XML |
| Obsługa wyjątków |  | Nie | tak | nie | tak | tak (Java) | Tak |

|  |  |  |
| --- | --- | --- |
| Technologia | Zalety | Wady |
| Gniazda (sockety) |  |  |
| RPC |  |  |
| RMI [xx] | - proste do wykonania  - wykorzystuje model bezpieczeństwa Javy  - interfejs Javy jest przedłużony do zdalnych obiektów  - dynamiczne tworzenie interfejsu | - technologia może działać wolniej niż inne alternatywne (zwłaszcza CORBA)  - język scentralizowany  - kod RMI jest wydajniejszy w języku Java |
| webServices |  |  |
| CORBA [xx] | - technologia szybsza od RMI  - pozwala na używanie różnych platform systemowych  - znacznie bardziej wszechstronna |  |
| SOAP | - elastyczność protokołu  - możliwość łączenia z różnymi protokołami transportowymi  - akceptowalność protokołu przez większość systemów i środowisk komputerowych  - niezawodność protokołu | - duży narzut języka XML  - duża ilość modyfikacji |
| XML |  |  |
| WSDL [xxx] | - standardowy format opisu usług  - umożliwia wykorzystanie różnych technologii i systemów komputerowych  - łatwy w użyciu  - automatyczna komunikacja między usługami  - prosty interfejs definiowania usług  - dynamiczny rozwój | - problemy z bezpieczeństwem  - problemy z autoryzacją  - mała wydajność  - dosyć częste zmiany |

[xx] <http://cs.iupui.edu/~aharris/cgi-bin/slides/langs0.html>

[xxx] http://galaxy.uci.agh.edu.pl/~ewao/referaty/WebServices.pdf

Trudności implementacyjne

- socket: różne biblioteki do obsługi gniazd na różnych systemach operacyjnych. W przypadku systemu Linux obsługa gniazd jest wbudowana, natomiast system Windows wymaga biblioteki WinSock.

- wsdl: zaimplementowanie dodatkowych protokołów bezpieczeństwa oraz autoryzacji

- CORBA: trudny w zrozumieniu interfejs, złożona struktura technologii

## Analiza porównawcza implementacji

|  |  |
| --- | --- |
|  |  |
| TECHNOLOGIA: |  |
| JĘZYK: |  |
| IMPLEMENTACJA DZIAŁA POPRAWNIE NA X KOMPUTERACH |  |
| IMPLEMENTACJA WYKRYWA PODSTAWOWE AWARIE: |  |
| IMPLEMENTACJA MA OPCJĘ TRWAŁEJ KONFIGURACJI USTAWIEŃ: |  |
| IMPLEMENTACJA POSIADA DOKUMENTACJĘ: |  |
| … |  |
|  |  |

## Wnioski

# Bibliografia (oczywiście trzeba będzie ją poprawić)

1. []. Ben-Ari M., „Podstawy programowania współbieżnego i rozproszonego, wyd. 2 zm. i rozsz.”, rozdział 12, str. 237-240, Wydawnictwa Naukowo-Techniczne, 2009, Warszawa [↑](#endnote-ref-2)
2. []. The Byzantine Generals Problem (with Marshall Pease and Robert Shostak), ACM Transactions on Programming Languages and Systems 4, 3 July 1982, <http://research.microsoft.com/users/lamport/pubs/byz.pdf>, (dostęp: 22.05.2016) [↑](#endnote-ref-3)
3. [].Berman P., Garay J A., Perry K.J., “Towards Optimal Distributed Consensus”, 30th IEEE Annual Symposium on Foundations of Computer Science (FOCS), str. 410-415, IEEE, ISBN: 0‑8186-1982-1, <http://zoo.cs.yale.edu/classes/cs426/2013/bib/berman89towards.pdf>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-4)
4. [].Ben-Or M., “Another advantage of free choice: completely asynchronous agreement protocols”, PODC ’83 Proceedings of the second annual ACM symposium on Principles of distributed computing, str.27-30, ACM New York, ISBN: 0-89791-110-5, <http://dl.acm.org/citation.cfm?id=806707>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-5)
5. [].Akkaoyunlu E.A., Ekanadham K., Huber R.V., “Some constraints and tradeoffs in the design of network communucations”, SOSP ’75 Proceedings of the fifth ACM symposium on Operating system principles, str. 67-74, ACM New York, DOI: 10.1145/800213.806523, <http://hydra.infosys.tuwien.ac.at/teaching/courses/AdvancedDistributedSystems/download/1975_Akkoyunlu,%20Ekanadham,%20Huber_Some%20constraints%20and%20tradeoffs%20in%20the%20design%20of%20network%20communications.pdf>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-6)
6. [].Berman P., Garay J.A., Perry K.J., “Optimal early stopping in distributed consensus”, 6th International Workshop, WDAG ’92, Haifa, Israel, Springer Berlin Heidelberg, ISBN: 978-3-540-56188-0, <http://link.springer.com/chapter/10.1007/3-540-56188-9_15>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-7)
7. [].Ongaro D., Ousterhout J., “In Search of an Understandable Consensus Algorithm”, Proc ATC’14 USENIX Annual Technical Conference, USENIX, <https://raft.github.io/raft.pdf>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-8)
8. []. Kumar A., “Performance analysis of a hierarchical quorum consensus algorithm for replicated objects”, str. 378-385, Distributed Computing Systems, 1990,ISBN: 0-8186-2048-X, <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=89306&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D89306>, (dostęp: 07.06.2016r.) [↑](#endnote-ref-9)
9. [].De Prisco R., Malkhi D., Reiter M.K., “On k-set Consensus Problems in Asynchronous Systems”, PODC ’99 Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, str. 257-265, ACM New York, ISBN: 1-58113-099-6, <https://www.cs.unc.edu/~reiter/papers/1999/PODC.pdf>, (dostęp: 03.06.2016r.) [↑](#endnote-ref-10)
10. []. Sape Mullender, Distributed Systems, Addison-Wesley; 2 edition (June 30, 1993),ISBN-13: 978-0201624274 [↑](#endnote-ref-11)
11. [].Aguilera M.K., Toueg S., “Correctness Proof of Ben-Or’s Randomized Consensus Algorithm”, Proceedings of the 10th International Workshop on Distributed Algorithms, Springer-Verlag, <http://disi.unitn.it/~montreso/ds/syllabus/papers/AguileraToeug-CorrecnessBenOr.pdf>, (dostęp: 04.06.2016r.) [↑](#endnote-ref-12)
12. [].Coulouris G., Dollimore J., Kindberg T., Gordon B., “Distributed Systems. Concepts and Design. Fifth Edition”, str. 662, Pearson Education, 2012, Boston [↑](#endnote-ref-13)
13. []. Mads Dam, “Parallel and Distributed Computing. Distributed Algorithms. Lecture 7: Consensus”, slajdy 72-82 oraz 93, <http://www.csc.kth.se/utbildning/kth/kurser/DD2451/pardis11/DD2451_lecture7.pdf>, (dostęp: 26.05.2016) [↑](#endnote-ref-14)
14. [].Nelson M., “The Byzantine Generals Problem”, artykuł: <http://marknelson.us/2007/07/23/byzantine/>, (dostęp: 22.05.2016) [↑](#endnote-ref-15)
15. []. “Systemy Równoległe i rozproszone”, wykład nr 13 – „Rozgłaszanie i uzgadnianie”, strona z materiałami dydaktycznymi dr hab. inż. K. Banasia, <http://www.metal.agh.edu.pl/~banas/SRR/SRR_W13_Rozglaszanie_Uzgadnianie.pdf>, (dostęp: 22.05.2016) [↑](#endnote-ref-16)
16. []. Ajay Kschemkalyani, Mukesh Singhal, „Distributed Computing: Principles, Algorithms and Systems. Chapter 14: Consensus and Agreement”, Cambridge University Press, <https://www.cs.uic.edu/~ajayk/Chapter14.pdf>(dostęp: 22.05.2016) [↑](#endnote-ref-17)
17. []. Mads Dam, “Parallel and Distributed Computing. Distributed Algorithms. Lecture 7: Consensus”,slajdy 48-55, <http://www.csc.kth.se/utbildning/kth/kurser/DD2451/pardis11/DD2451_lecture7.pdf>, (dostęp: 22.05.2016) [↑](#endnote-ref-18)
18. []. „Computer Science & Engineering. Courses. CSEP552: Distributed Systems.Lecture 5: Intro to consensus”, University of Washington, <http://courses.cs.washington.edu/courses/csep552/13sp/lectures/5/intro.pdf> (dostęp 21.05.2016) [↑](#endnote-ref-19)
19. []. Paul Krzyzanowski, „Consensus. Reaching agreement”, <https://www.cs.rutgers.edu/~pxk/417/notes/content/consensus.html>(dostęp: 22.05.2016) [↑](#endnote-ref-20)
20. [].<https://www.cs.uic.edu/~ajayk/Chapter14.pdf> (dostęp 26.05.2016) [↑](#endnote-ref-21)
21. []. <https://www.cs.unc.edu/~reiter/papers/1999/PODC.pdf> (dostęp 26.05.2016) [↑](#endnote-ref-22)