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** 3](#_Toc452803188)

[**2.** **Algorytmy rozwiązujące zagadnienie uzgadniania** 3](#_Toc452803189)

[**2.1.** **Algorytm bizantyjskich generałów** 4](#_Toc452803190)

[**2.2.** **Algorytm randomizowany** 4](#_Toc452803191)

[2.3. Algorytm jednorundowy 5](#_Toc452803192)

[2.4. Algorytm króla 5](#_Toc452803193)

[2.5. Algorytm królowej 5](#_Toc452803194)

[2.6. Algorytm dwóch armii 5](#_Toc452803195)

[2.7. k-set Consensus 5](#_Toc452803196)

[2.8. Terminating Reliable Broadcast 5](#_Toc452803197)

[2.9. Algorytm tratwy 5](#_Toc452803198)

[2.10. Rozgłoszeniowy algorytm konsensusu podstawowego 5](#_Toc452803199)

[2.11. Hierarchiczny algorytm konsensusu podstawowego 5](#_Toc452803200)

[2.12. Zestawienie najważniejszych informacji o wybranych algorytmach 5](#_Toc452803201)

[**3.** **Analiza porównawcza algorytmów** 7](#_Toc452803202)

[**4.** **Opis sposobu oceniania algorytmów oraz wybór najlepszego (?)** 7](#_Toc452803203)

[**5.** **Technologie rozproszone dostępne na rynku** 7](#_Toc452803204)

[**6.** **Analiza porównawcza implementacji** 8](#_Toc452803205)

[**7.** **Wnioski** 8](#_Toc452803206)

[**8.** **Bibliografia (oczywiście trzeba będzie ją poprawić)** 9](#_Toc452803207)

# **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-1)].

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

//MOŻNA JESZCZE COŚ DOPISAĆ

# **Algorytmy rozwiązujące zagadnienie uzgadniania**

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

## **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 [[[10]](#endnote-10)].

## Algorytm jednorundowy

## Algorytm króla

## Algorytm królowej

## Algorytm dwóch armii

## k-set Consensus

## Terminating Reliable Broadcast

## Algorytm tratwy

## Rozgłoszeniowy algorytm konsensusu podstawowego

## Hierarchiczny algorytm konsensusu podstawowego

## Zestawienie najważniejszych informacji o wybranych algorytmach

|  |  |  |  |
| --- | --- | --- | --- |
|  | Algorytm generałów bizantyjskich (The Byzantine Generals Problem) [[[11]](#endnote-11)] | Algorytm randomizowany (Randomized algorithm) [[[12]](#endnote-12)] | 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 pj są 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) [[[13]](#endnote-13)] | 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 [1] | ?? | 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 [[[14]](#endnote-14)] | 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) |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Algorytm króla (Phase King Algorithm)[[[15]](#endnote-15)] | Algorytm królowej (The Queen Algorithm)[[[16]](#endnote-16)] | 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.[[[17]](#endnote-17)] |
| 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[[[18]](#endnote-18)] |
| 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[1] | =(n-f)/2+(n-f)/2 > n/2+f wiadomości | Ilość wiadomości rośnie wykładniczo |
| złożoność obliczeniowa | O(t^3) | ??? | ??? |

|  |  |  |
| --- | --- | --- |
|  | Terminating Reliable Broadcast[[19]](#endnote-19) | k-set Consensus[[20]](#endnote-20) |
| 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). |  |

# **Analiza porównawcza algorytmów**

# **Opis sposobu oceniania algorytmów oraz wybór najlepszego (?)**

# **Technologie rozproszone dostępne na rynku**

Obecnie na rynku dostępnych jest wiele technologii rozproszonych. Do tych najważniejszych zaliczają się:

* gniazda (sockety)
* RPC
* RMI
* webServices

Ponadto: CORBA, SOAP, WSDL

# **Analiza porównawcza implementacji**

# **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-1)
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-2)
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-3)
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-4)
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-5)
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-6)
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-7)
8. 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-8)
9. Sape Mullender, Distributed Systems, Addison-Wesley; 2 edition (June 30, 1993), ISBN-13: 978-0201624274 [↑](#endnote-ref-9)
10. 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-10)
11. Coulouris G., Dollimore J., Kindberg T., Gordon B., “Distributed Systems. Concepts and Design. Fifth Edition”, str. 662, Pearson Education, 2012, Boston [↑](#endnote-ref-11)
12. 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-12)
13. Nelson M., “The Byzantine Generals Problem”, artykuł: <http://marknelson.us/2007/07/23/byzantine/>, (dostęp: 22.05.2016) [↑](#endnote-ref-13)
14. “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-14)
15. 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-15)
16. 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-16)
17. „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-17)
18. Paul Krzyzanowski, „Consensus. Reaching agreement”, <https://www.cs.rutgers.edu/~pxk/417/notes/content/consensus.html> (dostęp: 22.05.2016) [↑](#endnote-ref-18)
19. <https://www.cs.uic.edu/~ajayk/Chapter14.pdf> (dostęp 26.05.2016) [↑](#endnote-ref-19)
20. <https://www.cs.unc.edu/~reiter/papers/1999/PODC.pdf> (dostęp 26.05.2016) [↑](#endnote-ref-20)