Bazy danych

11. Systemy rozproszone i twierdzenie CAP

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2019

Nowe wyzwania

Sytuacja na przełomie lat dziewięćdziesiątych i dwutysięcznych:

- duże wolumeny danych (z czasem: Big Data)
- konieczność stosowania dynamicznych schematów baz danych
- konieczność replikacji...
- ...i skalowania poziomego
- a wszystko to za pośrednictwem sieci

Eric Brewer, 2000: Mamy strukturalny problem... ☹
Problem ten manifestuje się tylko dla *naprawdę dużych* wolumenów danych i *naprawdę dużego* ruchu.

Eric Brewer

Fallacies of Distributed Computing

(Patrz ten TechTalk)

- Zawsze można polegać na sieci
- Nie występują opóźnienia
- Przepustowość jest nieograniczona
- Połączenia sieciowe są bezpieczne
- Topologia sieci się nie zmienia
- Jest tylko jeden administrator
- Przesyłanie danych jest bezkosztowe
- Sieć jest jednorodna

Podstawowe pojęcia — spójność

Spójność (consistency) baz danych — dane na wszystkich serwerach przechowujących kopie danej bazy są identyczne (i spełniają więzy narzucone na bazę).

Pojęcie to można uogólnić na dowolne usługi dostępne przez sieć (wyszukiwarki, DNS*, e-sklepy, serwisy webowe itd): Spójność oznacza, że istnieje gwarancja, iż każdy odczyt zwróci wynik najnowszego zapisu danej wartości. Daje się to zrealizować, jeżeli operacje na danych są serializowalne: *Jeśli operacja* B *rozpoczęła się po tym, gdy operacja* A *prawidłowo się zakończyła, wówczas operacja* B *musi widzieć taki stan systemu, jaki był w momencie zakończenia* A*, lub późniejszy.*

Bazy OLTP realizują spójność na przykład poprzez protokół 2PC.

*Tak jest w teorii. W praktyce systemy DNS oparte na kaszowaniu zapewniają dużą dostępność, ale słabą spójność — patrz niżej.

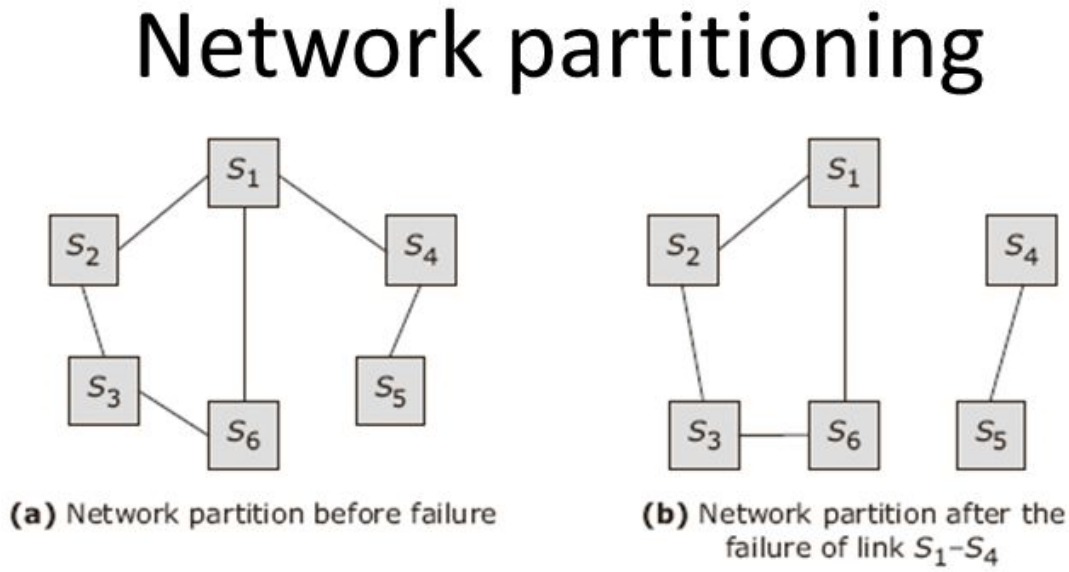
Podstawowe pojęcia — dostępność

Dostępność (availability) — istnieje gwarancja, że każde zapytanie zwróci wiarygodną odpowiedź w skończonym czasie, bez utraty komunikatów, bez konieczności zgłaszania błędu, w tym błędu typu *timeout*.

Szybka odpowiedź jest lepsza od braku odpowiedzi, ale w tym kontekście wystarcza, aby usługa *kiedyś, w końcu* (ang. *eventually*) udzieliła wiaygodnej odpowiedzi.

Istnieje także pojęcie *eventual consistency*: Dopuszczamy, że system przez jakiś czas nie będzie spójny, ale osiągnie spójność po dostatecznie długim czasie.

Odporność na partycjonowanie (partition tolerance) — istnieje gwarancja, że system będzie funkcjonował także po spartycjonowaniu sieci. Innymi słowy, system zawiedzie dopiero gdy wszystkie węzły zawiodą.



(a) Network partition before failure

(b) Network partition after the failure of link S_1–S_4

Twierdzenie CAP

Rozproszona baza danych — a szerzej, dowolny rozproszony system obliczeniowy — może jednocześnie spełnić co najwyżej dwa z trzech wymagań: spójność, dostępność i odporność na partycjonowanie.

Eric Brewer, 2000

Twierdzenie CAP uznawane jest za prawdziwe (istnieją formalne dowody tego twierdzenia), choć jak każde twierdzenie mówiące o niemożliwości jakiegoś zjawiska (stanu), często jest kontestowane — patrz na przykład blog Marka Burgessa. (Kontestacja polega wówczas na kwestionowaniu założeń, tego, czy formalne założenia prawidłowo odzwierciedlają rzeczywistość.)

W nieco bardziej nowoczesnym sformułowaniu, twierdzenie CAP brzmi:

W sieci podatnej na błędy komunikacji, nie jest możliwe, aby jakikolwiek system sieciowy zapewniał w każdych warunkach spójność danych i jednocześnie to, że każde zapytanie doczeka się odpowiedzi.

Konsensus

Nieco abstrakcyjną ideę *spójności* zastępuje się pojęciem *konsensusu*. Jeśli mamy procesy $\{p_i\}_{i=1}^{N}$, z których każdy zwraca pewną wartość v_i, wszystkie poprawne (prawidłowo przebiegające, nie zakończone błędem) procesy muszą zgodzić się odnośnie do wspólnego wyniku, przy czym

- Zgodność: Każdy poprawny proces musi zgodzić się na ten sam wynik v.
- Prawidłowość (ang: *validity*): Jeżeli wszystkie poprawne procesy proponują tę samą wartość v, to wszystkie poprawne procesy zgadzają się na tę samą wartość v.
  - Słaba prawidłowość: Jeżeli wszystkie poprawne procesy otrzymują takie same dane wejściowe, wszystkie muszą zwrócić tę samą wartość wynikową v.

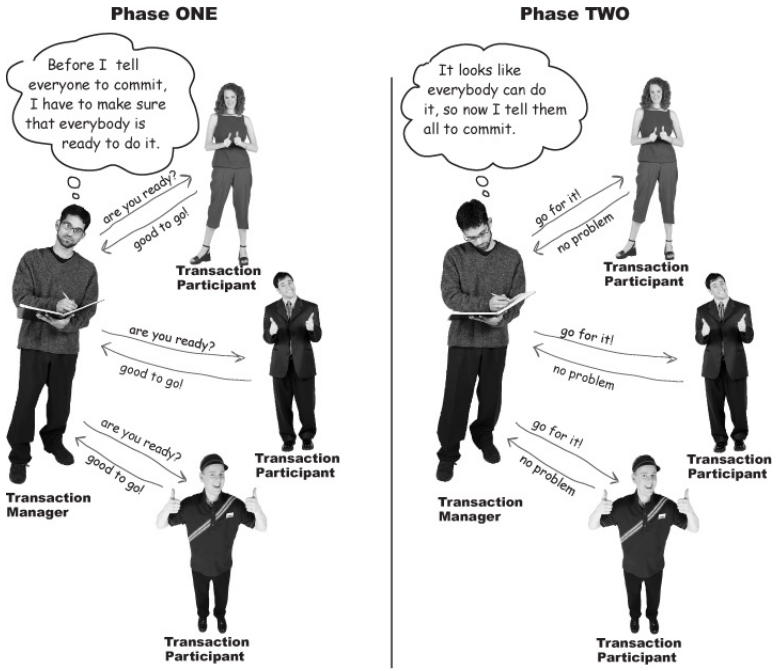
  - Silna prawidłowość: Dla każdego poprawnego procesu, jego wartość wynikowa musi stanowić wartość wejściową jakiegoś poprawnego procesu.
- Warunek zakończenia (ang. *termination*): Każdy prawidłowy proces musi w końcu zwrócić jakąś wartość.

Konsensus może być osiągany na różne sposoby, na przykład w drodze głosowania, lub w inny sposób, pozwalający uzyskać “wspólne zdanie” pomiędzy procesami (węzłami sieci).

Prawidłowo zakończony proces 2PC jest także przykładem konsensusu.

Przypomnienie - protokół *Two-phase commit*

Wszystkie komputery w sieci muszą się zgodzić na pewne działanie. Jeśli któryś się nie zgodzi lub nie odpowie w określonym czasie, operacja zostaje odwołana



Byzantine Generals Problem

Wednesday, August 18, 2010

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, July 1982, Pages 382–401.

A reliable computer system must be able to cope with the failure of one or more of its components. A failed component may exhibit a type of behavior that is often overlooked—namely, sending conflicting information to different parts of the system. The problem of coping with this type of failure is expressed abstractly as the Byzantine Generals Problem. We devote the major part of the paper to a discussion of this abstract problem and conclude by indicating how our solutions can be used in implementing a reliable computer system.

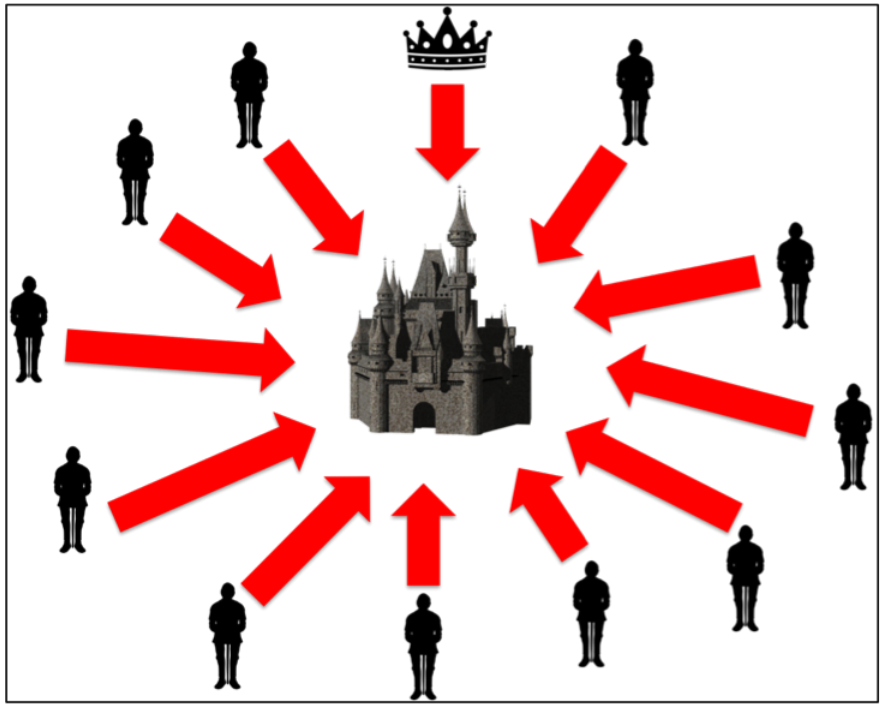
This research was supported in part by the National Aeronautics and Space Administration under contract NAS1-15428 Mod. 3, the Ballistic Missile Defense Systems Command under contract DASG60-78-C-0046, and the Army Research Office under contract DAAG29-79-C-0102.
Authors' address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025.

Katastrofa bizantyńska

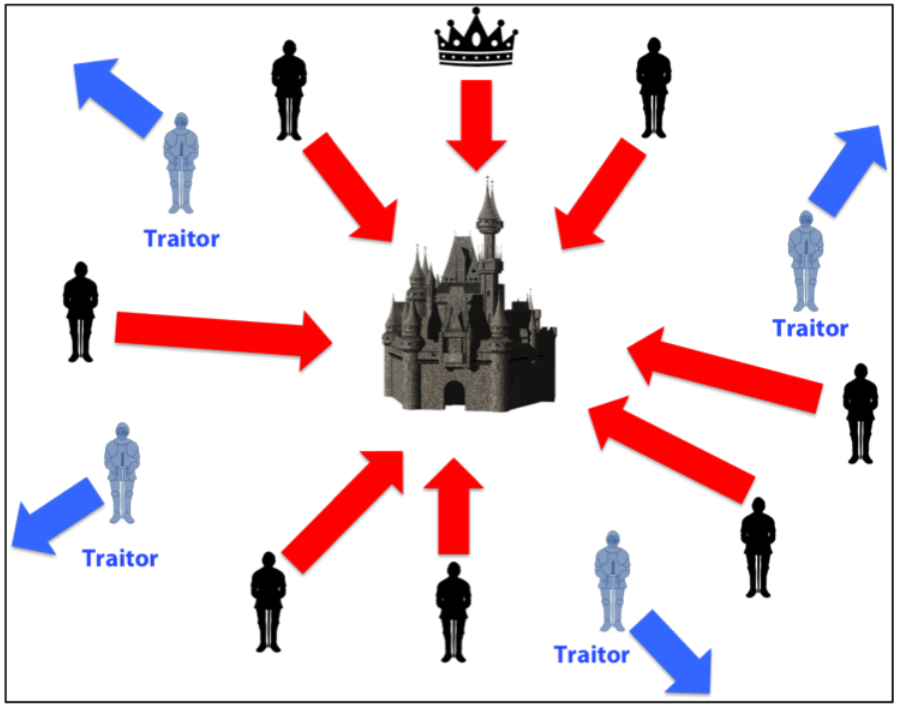
Problem bizantyńskich generałów: Kilka armii oblega miasto. Ich dowódcy muszą się zgodzić na wspólny atak lub wspólny, zorganizowany odwrót, gdyż to daje szanse na zwycięztwo. Sytuacja, w której część oddziałów atakuje, część się wycofuje, prowadzi do porażki.

Generałowie muszą osiągnąć konsensus odnośnie do strategii.

Wśród generałów są jednak zdrajcy, którzy przekazują niewiarygodne komunikaty, w dodatku różne dla różnych odbiorców. Na przykład dowódcom, którzy chcą atakować, zdrajca mówi, że chce się wycofać, a dowódcom, którzy chcą się wycofać, zdrajca mówi, że chce atakować.



Coordinated Attack Leading to Victory

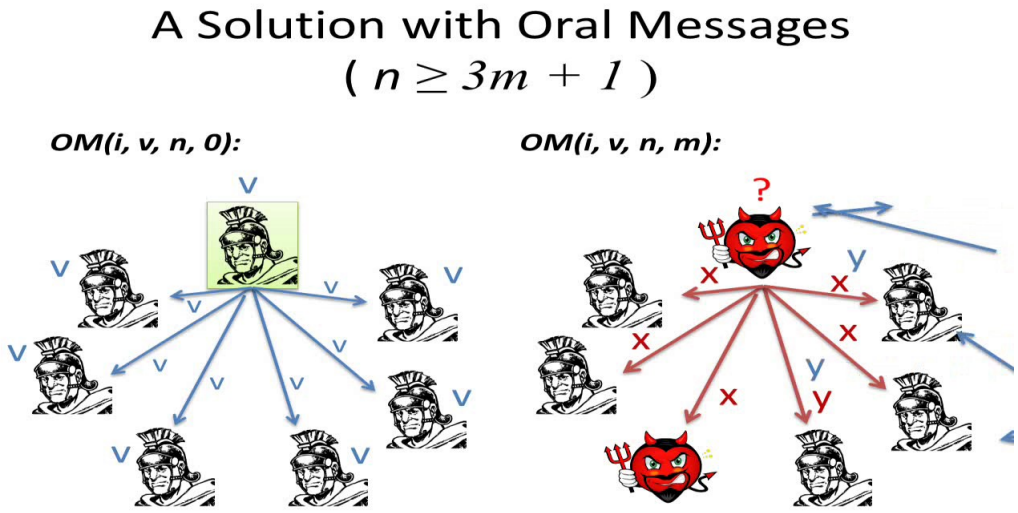


Uncoordinated Attack Leading to Defeat

W kontekście bazodanowym — czy ogólniej, w kontekście systemów rozproszonych — oznacza to sytuację, w której jeden węzeł zawodzi i część pozostałych węzłów widzi, że działa on nieprawidłowo (więc jego opinię należy zignorować), ale część twierdzi, że działa on poprawnie (więc jego opinii nie można ignorować). Nazywa się to *katastrofą bizantyńską* i oznacza sytuację, w której różni obserwatorzy różnie oceniają prawidłowość działania danego węzła.

W historii systemów rozproszonych **zdarzyło się** kilka dobrze udokumentowanych katastrof bizantyńskich.

W języku sieci komputerowych: Czy uda się uzgodnić *zadowalające* rozwiązanie (konsensus), jeśli pewna liczba komputerów działa nieprawidłowo (awaria, atak hakerów)?



Klasyczne rozwiązanie: Ponad 2/3 komputerów musi działać prawidłowo.

Sieci synchroniczne i asynchroniczne

Definicja: Sieć jest synchroniczna, jeżeli (a) każdy proces ma zegar, a wszystkie zegary są zsynchronizowane, (b) każda wiadomość rozsyłana jest w stałym i znanym czasie, (c) każdy proces przebiega w stałym i znanym tempie.

Taka sieć działa w rundach (obejściach): W czasie jednej rundy każdy proces wysyła pewną liczbę wiadomości, odbiera wszystkie wiadomości, które zostały do niego wysłane, a także przeprowadza lokalne obliczenia.

Sieć, która nie jest synchroniczna, jest nazywana asynchroniczną.

Wyniki formalne

Istnieje szereg formalnych wyników związanych z twierdzeniem CAP, a ściślej, z pojęciem konsensusu.

- W sieciach asynchronicznych uzyskanie konsensusu jest niemożliwe, jeśli choć jeden węzeł (proces) zawiedzie.
- W sieciach synchronicznych, w których co najwyżej f węzłów zawiedzie (w których istnieje co najwyżej f nieprawidłowych procesów), kosensus można uzyskać po nie wiećej niż $f+1$ rundach.
- k-*set agreement*: sieć zgadza się na k różnych wyników. 1-*set agreement* jest równoważny konsensusowi, w którym żaden węzeł (proces) nie zawodzi. k-*set agreement* może zostać osiągnięty jeżeli co najwyżej $k-1$ węzłów (procesów) zawiedzie. Można powiedzieć, że k-*set agreement* jest "najlepszym" stopniem zgodności, jaki można osiągnąć dopuszczając, że $k-1$ węzłów (procesów) zawiedzie.

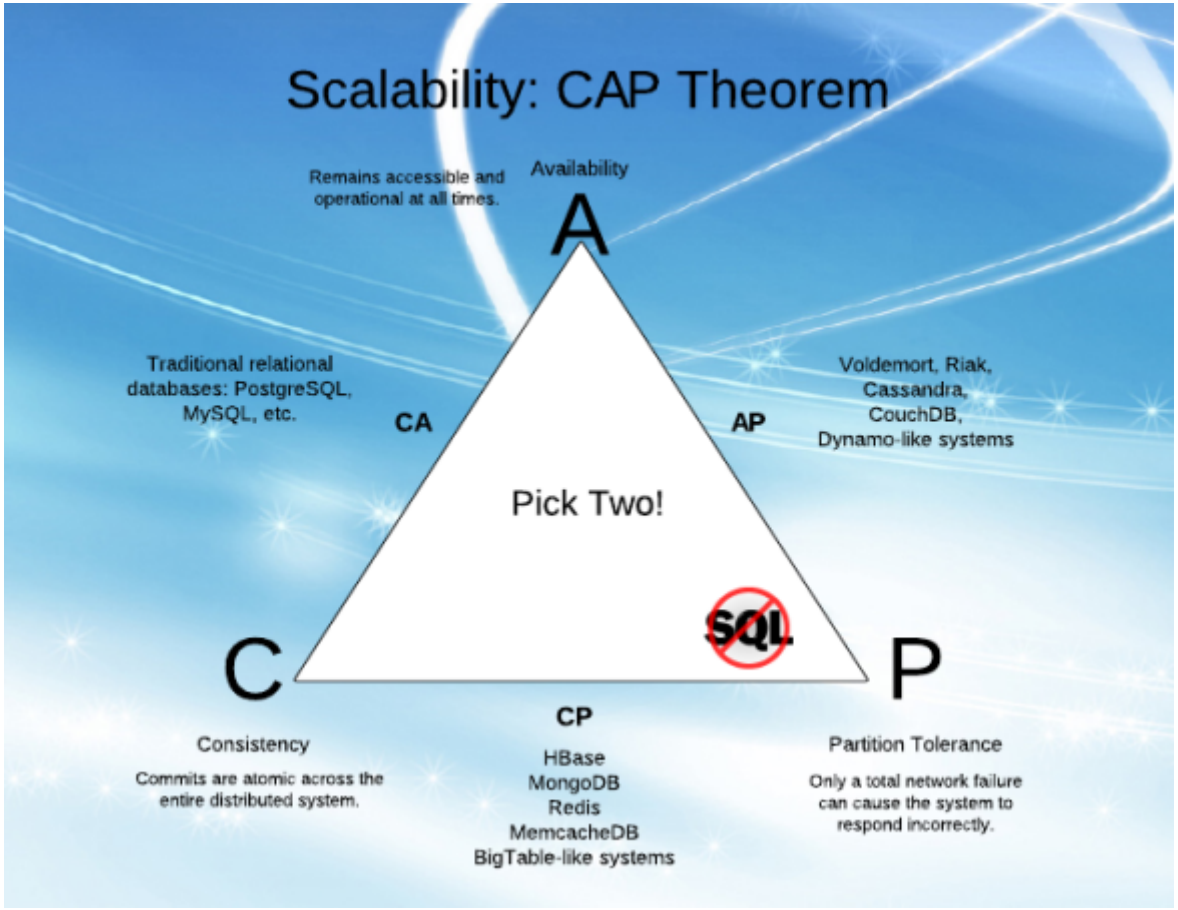
Zawodna analogia

Można pomyśleć, że twierdzenie CAP brzmi trochę jak stary kawał:

Nasza firma oferuje usługi szybkie, tanie i dobre. . .

. . . ale zapewnia jednoczesne spełnienie co najwyżej dwu z tych wymagań

- Jeśli ma być szybko i tanio, to nie będzie dobrze.
- Jeśli ma być szybko i dobrze, to nie będzie tanio.
- Jeśli ma być tanio i dobrze, to nie będzie szybko.



Analogia ta jest jednak zwodnicza! W rzeczywistoście jest jeszcze gorzej.

Należy zakładać, że każda rzeczywista sieć może zawieść.

Nie ma niezawodnych sieci. To, że sieć zawodzi, jest niezależne od projektanta oprogramowania. Wydaje się zatem, że projektując oprogramowanie, możemy wybierać jedynie pomiędzy dwoma (a nie trzema!) rozwiązaniami:

CP Consistency/Partition Tolreance, zapewniając *najlepszą możliwą* dostępność: Czekaj na odpowiedź od niedostępnego na skutek spartycjonowania sieci węzła, co może spowodować błąd typu *timeout*, oznaczający brak dostępności. To rozwiązanie wybiera się, jeżli najważniejszym warunkiem jest atomowość operacji.

AP Availability/Partition Tolerance, zapewniając *najlepszą możliwą* spójność: Jeśli otrzymasz zapytanie, odpowiedz przesyłając najbardziej aktualne

dane, które posiadsz, ryzykując, że mogą one być przestarzałe, bo na aktualnie niedostępnym węźle ktoś je zmodyfikował. Tę opcję wybiera się, jeśli system musi funkcjonować pomimo wystąpienia zewnętrznych błędów.

To samo inaczej

Jeżeli wystąpi błąd komunikacji (błąd sieci, partycjonowanie sieci):

CP Jeśli system nie może zapewnić, że dostarcza najbardziej aktualną informację, przestaje odpowiadać.

AP System cały czas odpowiada, mimo iż (niektóre) dostarczane przez niego dane mogą być nieaktualne.

Inne możliwe rozwiązania

Partycjonowanie danych. Różne rodzaje danych mogą wymagać różnych stopni spójności i dostępności, skoro nie mogą osiągnąć jednocześnie jednego i drugiego. Na przykład w e-sklepie “koszyk” musi być stale dostępny, musi reagować na żądania użytkownika, ale w niektórych wypadkach może tracić informację o ostatnich działaniach użytkownika: jest niespójny, ale jeżeli niespójność trwa krótko, nie spowoduje znacznego dyskomfortu użytkownika. Niespójna może być też informacja o dostąpnych produktach (gdzieś pokazuje się informacja aktualna, gdzie indziej przestarzała, na przykład z ceną sprzed ogłoszenia promocji). Jednak ostateczny rachunek (zawartość zamówienia), koniecznie musi być spójny: użytkownicy nie zgodzą się, aby dotarły do nich inne produkty, niż te, które faktycznie zamówili.

Partycjonowanie operacji. Niektóre operacje mogą wymagać innych parametrów dostępności i spójności. Na przykład oczekujemy, że operacje odczytu muszą być zawsze dostępne (nawet za cenę odczytania przestarzałych danych), ale operacje zapisu (modyfikacji) danych muszą być spójne. Gdybyśmy wymagali, że operacje odczytu *zawsze* muszą być dostępne, a operacje zapisu *zawsze* muszą być spójne, prowadziłoby to załamania systemu w wypadku awarii sieci. W praktyce zgadzamy się więc, że (niektóre) operacje zapisu stają się niedostępne w przypadku wystąpienia awarii sieci — na przykład protokół 2PC wycofuje (`rollback`) transakcję po wystąpieniu błędu *timeout*.

Partycjonowanie użytkowników. W systemach geograficznie rozległych, użytkownicy fizycznie najbardziej oddaleni od serwera mogą mieć największe trudności i z dostępnością, i z jakością (aktualnością) dostępnych danych. Replikujemy więc serwery, umieszczając je w różnych geograficznych lokalizacjach. Użytkownicy korzystają z najbliższego im serwera. Jest on dla nich dostępny i zapewnia spójność danych zapisanych na tym serwerze, jednak jeśli dane zostały zmodyfikowane przez użytkownika korzystającego z odległego serwera, spójność nie jest zapewniana — zachodzi jedynie *eventual consistency*.

Istnieją także inne modele, nie będące ani CP, ani AP, ale lepiej — z punktu widzenia danego serwisu — ustalające balans pomiędzy spójnością a dostępnością.

