# Thema Paxos

**2019-04-30**

**Autoren**

Marc-Antoine Brülhart

Carlo Baumann

Carole Hug

Simon Wächter

<https://en.wikipedia.org/wiki/Paxos_(computer_science)>

# Einleitung

Verteilte Systeme bestehen aus einigen separaten Geräten, was in der Regel als Vorteil eine hohe Verfügbarkeit und Skalierbarkeit bietet. Es kann selten davon ausgegangen werden, dass die einzelnen Geräte zuverlässig funktionieren. Sie gehen “down” und kommen wieder hoch und können unterschiedliche Ziele haben. Bei manchen Transaktionen ist es notwendig, dass sich die einzelnen Komponenten einigen, ob sie ausgeführt oder abgebrochen werden. Um dieses Problem zu lösen hat Leslie Lamport mit Fred Schneider 1989 den Paxos-Algorithmus entwickelt. Dieser sieht vor, dass die beteiligten Komponente unterschiedliche Rollen einnehmen und jeweils koordiniert darüber abstimmen, was mit diesen Transaktionen geschehen soll. Letztendlich durchgeführt wird das, was die Mehrheit entscheidet. Dieser Lösungsansatz ist zwar ziemlich komplex, aber eine ziemlich elegante Lösung

* Geschichte
* Was ist Consensus?
  + Beschreiben des Consensus Problems
* Was ist Paxos?
* Verwendungszweck

*The distributed transaction commit problem requires reaching agreement on whether a transaction is committed or aborted. The classic Two-Phase Commit protocol blocks if the coordinator fails. Fault-tolerant consensus algorithms also reach agreement, but do not block whenever any majority of the processes are working. The Paxos Commit algorithm runs a Paxos consensus algorithm on the commit/abort decision of each participant to obtain a transaction commit protocol that uses 2F + 1 coordinators and makes progress if at least F +1 of them are working properly. Paxos Commit has the same stable-storage write delay, and can be implemented to have the same message delay in the fault-free case, as Two-Phase Commit, but it uses more messages. The classic Two-Phase Commit algorithm is obtained as the special F = 0 case of the Paxos Commit algorithm.*

# Hauptteil

## Das Consensus Problem (2 Phase Commit, Paxos)

Eine Herausforderung verteilter Systeme ist trotz einzelner fehlerhafter Prozesse einen zuverlässigen Betrieb zu gewährleisten. Dazu ist es oft notwendig, dass sich die Prozesse untereinander auf einzelne Werte einigen, die für weitere Berechnungen verwendet werden. Mit dieser Fragestellung befasst sich das Consensus-Problem. Eine Lösung wäre, jeweils die beteiligten Prozesse abstimmen und die Mehrheit entscheiden zu lassen.

## Der Paxos Algorithmus

### Voraussetzungen

Damit der Paxos-Algorithmus fehlerfrei funktioniert, müssen folgende Voraussetzungen erfüllt sein:

* Alle beteiligten Prozesse müssen sich untereinander kennen.
* Alle beteiligten Prozesse müssen direkt mit allen anderen kommunizieren können.
* Die versendeten Nachrichten dürfen nie nachträglich manipuliert werden können.
* Byzantinische Fehler [[1]](#footnote-0)können ausgeschlossen werden.
* Die Daten der einzelnen Prozesse müssen so gespeichert werden, dass sie sich auch nach einem allfälligen Neustart nicht widersprüchlich verhalten.
* Die beteiligten Prozesse sind entweder vollumfänglich oder gar nicht verfügbar. Ein Zustand dazwischen darf nicht angenommen werden.
* Es muss definiert sein, wie festgestellt wird, ob die Mehrheit (= Konsensus??) erreicht wurde.

### Rollen

Im Paxos Protokoll werden Aktionen der Prozesse als Rollen beschrieben. In einer Paxos Implementation können die Prozesse mehrere Rollen besitzen. Dies verbessert die Latenz und die Anzahl Nachrichten im Protokoll. Die Korrektheit des Protokolls wird dadurch nicht beeinflusst. Die wichtigsten Rollen sind Client, Proposer und Acceptor.

#### Client

Der Client ist nicht Teil des Paxos Systems. Er sendet Anfragen an das System. Anfragen (Request) werden immer durch einen Client ausgelöst und durch eine Antwort (Response) vom Paxos System abgeschlossen.

#### Proposer

Der Proposer empfängt die Anfrage vom Client und eröffnet eine Wahl im System. Er versucht eine Mehrheit von den Acceptors zu erhalten.

#### Acceptor

Der Acceptor, auch Voter genannt, ist ein Teilnehmer im System. Die Mehrheit der Acceptoren entscheidet über eine Änderung.

#### Leader

Der Leader ist eine spezielle Form des Proposers. Ein Leader gewährleistet den Fortschritt. Dies ist jedoch nur möglich wenn es einen einzigen Leader gibt. Mehrere Leader können den Fortschritt verzögern. In den meisten Paxos Implementationen gibt es nur einen Leader.

#### Learner

Der Learner handelt sobald sich die Acceptors auf eine Anfrage geeinigt haben. Er führt die Anfrage aus und sendet die Antwort an den Client. Für eine bessere Verfügbarkeit können mehrere Learner bereitgestellt werden. Ein Learner dient als Replikationsfaktor im Paxos Protokoll.

### Paxos

*Paxos unterscheidet zwei Ansätze wie Paxos angewendet werden kann. Dies sind Basic-Paxos (“single decree”) und Multi-Paxos. Multi-Paxos ist im wesentlichen die Clusterung von mehreren Basic-Paxos Instanzen.*

### Der Algorithmus

Wie bereits Eingangs erwähnt, hat der Paxos Algorithmus das Ziel dass sich die Mehrheit der Acceptors sich auf einen Wert einigen. Dies kann Beispielsweise angewendet werden, wenn entschieden werden muss, welche Transaktion schreiben darf. Jeder Prozessor im Paxos System läuft auf einem eigenen Knoten (Node). Um sicherzustellen, dass die Einigung stattfinden kann, müssen nebst den allgemeinen Voraussetzungen folgende Bedingungen erfüllt sein:

* Ein Knoten funktioniert vollständig oder nicht.
* Nachrichten werden vollständig oder nicht gesendet.
* Die Knoten haben einen persistenten Speicher

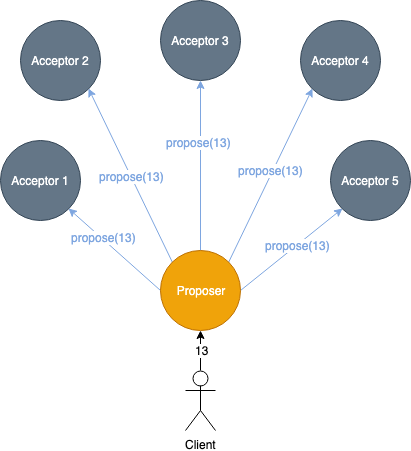
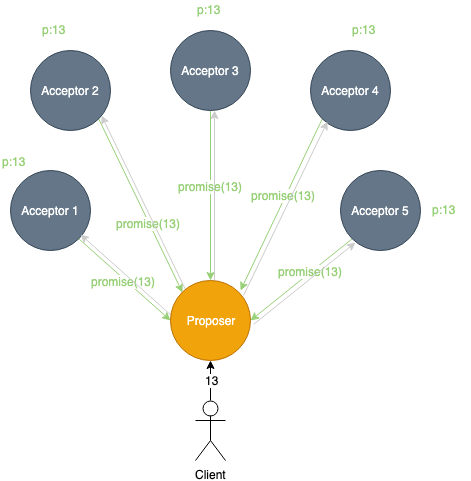
Die grundlegende Sicherheitsanforderung ist, dass nur ein Wert gewählt wird. Im Paxos System werden *2N+1* Acceptors benötigt um einen Consensus zu finden.

Der Paxos Algorithmus ist ein asynchroner Algorithmus. Der Algorithmus wird in zwei Phasen unterteilt.

#### Phase 1 - Prepare/Promise

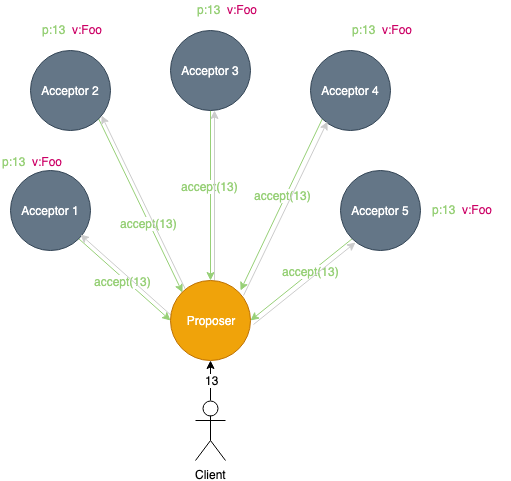
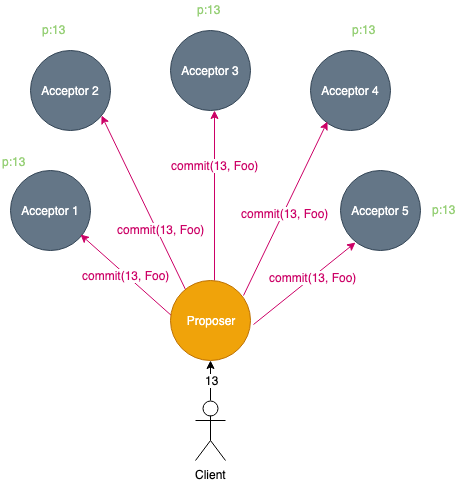
Ein Proposer sendet eine prepare(N) Anfrage an alle Acceptors. Die Anfrage muss mindestens an eine Mehrheit der Acceptors gesendet werden, um wiederum eine Mehrheit zu erhalten. Die prepare-Anfrage muss einen numerischen Wert N, den Identifier, enthalten.

Jeder Acceptor überprüft nun ob der Wert aus der prepare-Anfrage grösser als die bisher erhaltenen Werte ist. Nach Lamport: “If the proposer receives the requested responses from a majority of the acceptors, then it can issue a proposal with number n and value v, where v is the value of the highest-numbered proposal among the responses or is any value selected by the proposer if the responders reported no proposals.” [[2]](#footnote-1) Ist diese Bedingung erfüllt wird ein Versprechen (Promise) an den Proposer gesendet. Die Acceptors merken sich den Wert, welchen sie versprochen haben.

#### Phase 2 - Commit/Accept

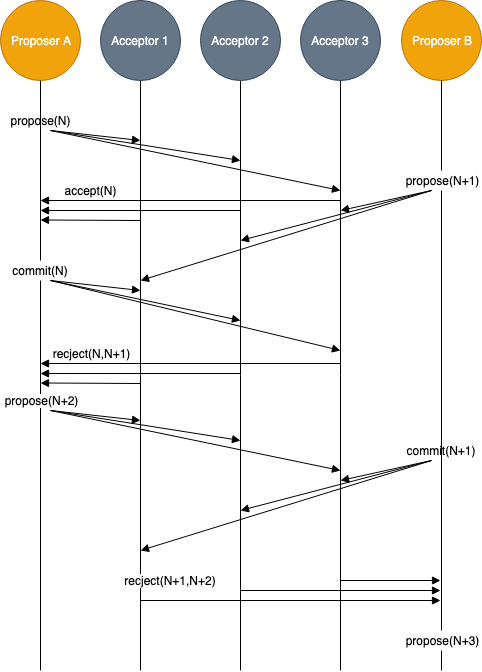
In der zweite Phase sendet der Proposer eine Commit-Anfrage an die Acceptors. Die Commit-Anfrage enthält den Identifier (N) und den Wert (V). Die Acceptors bestätigen diesen Wert und senden eine Accept-Antwort mit dem Identifier an den Proposer zurück. Die Acceptoren müssen sich die Werte für das vorgeschlagene N, das akzeptierte N und den akzeptierten Wert V auf einem persistenten und dauerhaften Speicher merken.



#### Fehlertoleranz

Das Paxos Protokoll ist viel fehlertoleranter als der 2-Phase-Commit. Dies weil bis zur Hälfte der Acceptors ausfallen können und dennoch eine Consensus gefunden werden kann. Darum müssen in einem Paxos System immer 2N+1 Teilnehmer sein. Fällt der Proposer aus, so kann ein anderer Acceptor die Rolle des Proposer übernehmen.

Wenn zwei Antragsteller gleichzeitig aktiv sind, können sie sich um die höchste Antragsnummer "duellieren". Dies machen die Proposer indem sie ihren Identifier um Eins erhöhen. Dies wird solange gemacht bis ein Leader erkoren wird. Solange diese Situation nicht gelöst ist, ist es möglich, dass der Paxos-Algorithmus nicht beendet wird. Es ist jedoch wahrscheinlich, dass Paxos zu einer korrekten Ausführung zurückkehrt, sobald sich das Netzwerk beruhigt hat und die beiden Antragsteller sich gegenseitig kennen und einem der Vortritt gelassen wird.



### Multi Paxos

Paxos versus Two-Phase Commit

Einsatzmöglichkeiten

* Datenbanken: Paxos Commit
* Filesysteme: Disk-Paxos
* Google: Chubby, Spanner
* Microsoft: Bing-Suchmachine

# Zusammenfassung

# Quellenangaben

[*https://www.the-paper-trail.org/post/2009-02-03-consensus-protocols-paxos/*](https://www.the-paper-trail.org/post/2009-02-03-consensus-protocols-paxos/)

<https://de.wikipedia.org/wiki/Paxos_(Informatik)#Rollen>

<https://lamport.azurewebsites.net/video/consensus-on-transaction-commit.pdf>

<http://www.goodmath.org/blog/2015/01/30/paxos-a-really-beautiful-protocol-for-distributed-consensus/>

<https://dbs.uni-leipzig.de/file/NoSQL_SS14_05_RecordStores.pdf>

In Search of an Understandable Consensus Algorithm (Extended Version) Diego Ongaro and John Ousterhout Stanford University

<https://ramcloud.stanford.edu/~ongaro/thesis.pdf>

<https://en.wikipedia.org/wiki/Paxos_(computer_science)>

<https://medium.com/coinmonks/paxos-made-simple-3b83c05aac37>

<https://blockonomi.com/paxos-raft-consensus-protocols/>

<http://www.goodmath.org/blog/2015/01/30/paxos-a-really-beautiful-protocol-for-distributed-consensus/>

1. <https://blockonomi.com/practical-byzantine-fault-tolerance/> [↑](#footnote-ref-0)
2. <https://blockonomi.com/paxos-raft-consensus-protocols/> [↑](#footnote-ref-1)