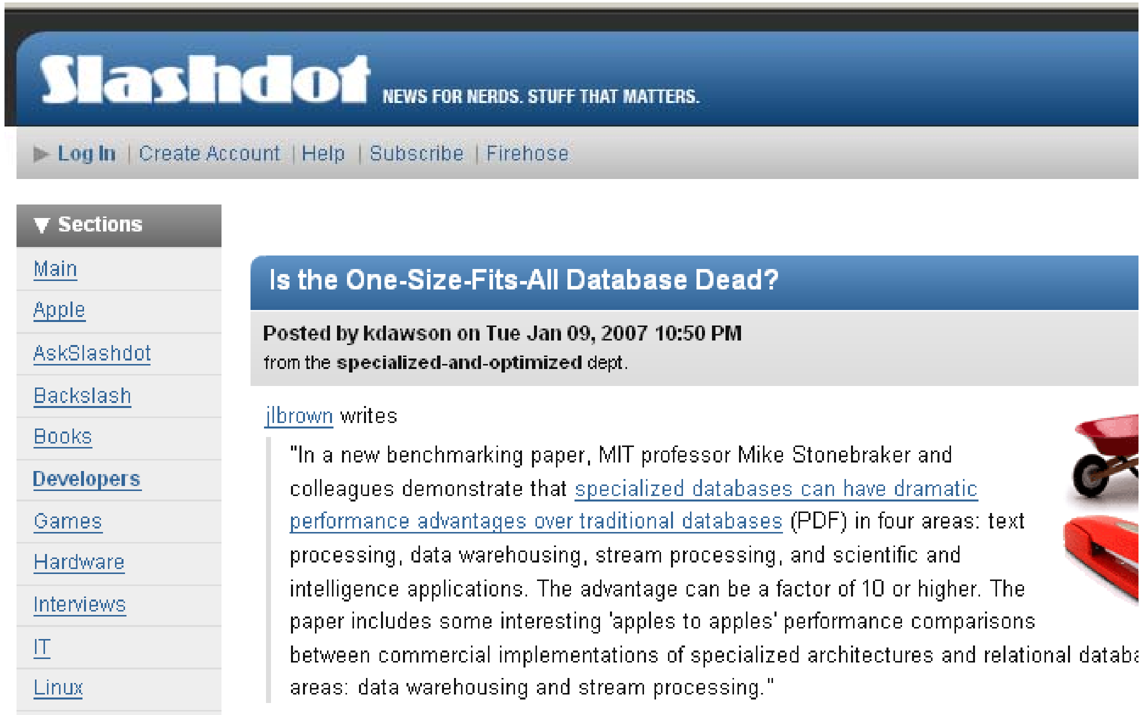
Building a transactional distributed data store with Erlang

Alexander Reinefeld, Florian Schintke, Thorsten Schütt

Zuse Institute Berlin,
onScale solutions GmbH

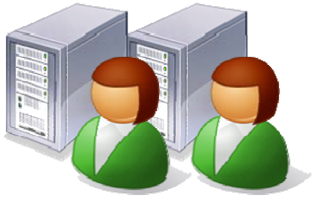
Transactional data store - What for?

- **Web 2.0 services**: shopping, banking, gaming, …
  - don’t need full SQL semantics, key/value DB often suffice
    e.g. Mike Stonebreaker: “One size does *not* fit all”



- **Scalability matters**
  - $>10^4$ accesses per sec.
  - many concurrent writes

Traditional Web 2.0 hosting



Clients

1 Server

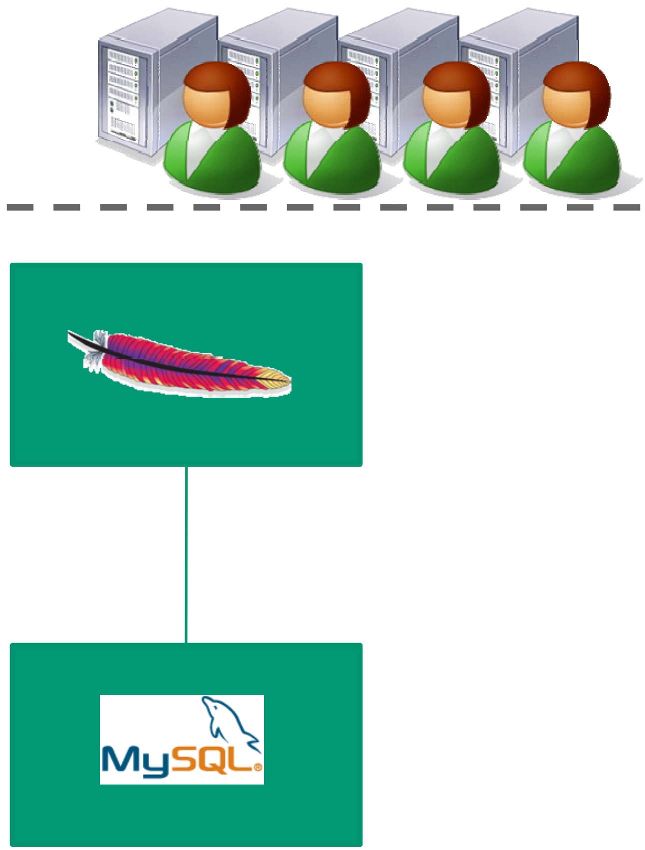
- Webserver
- DB Server

Traditional Web 2.0 hosting

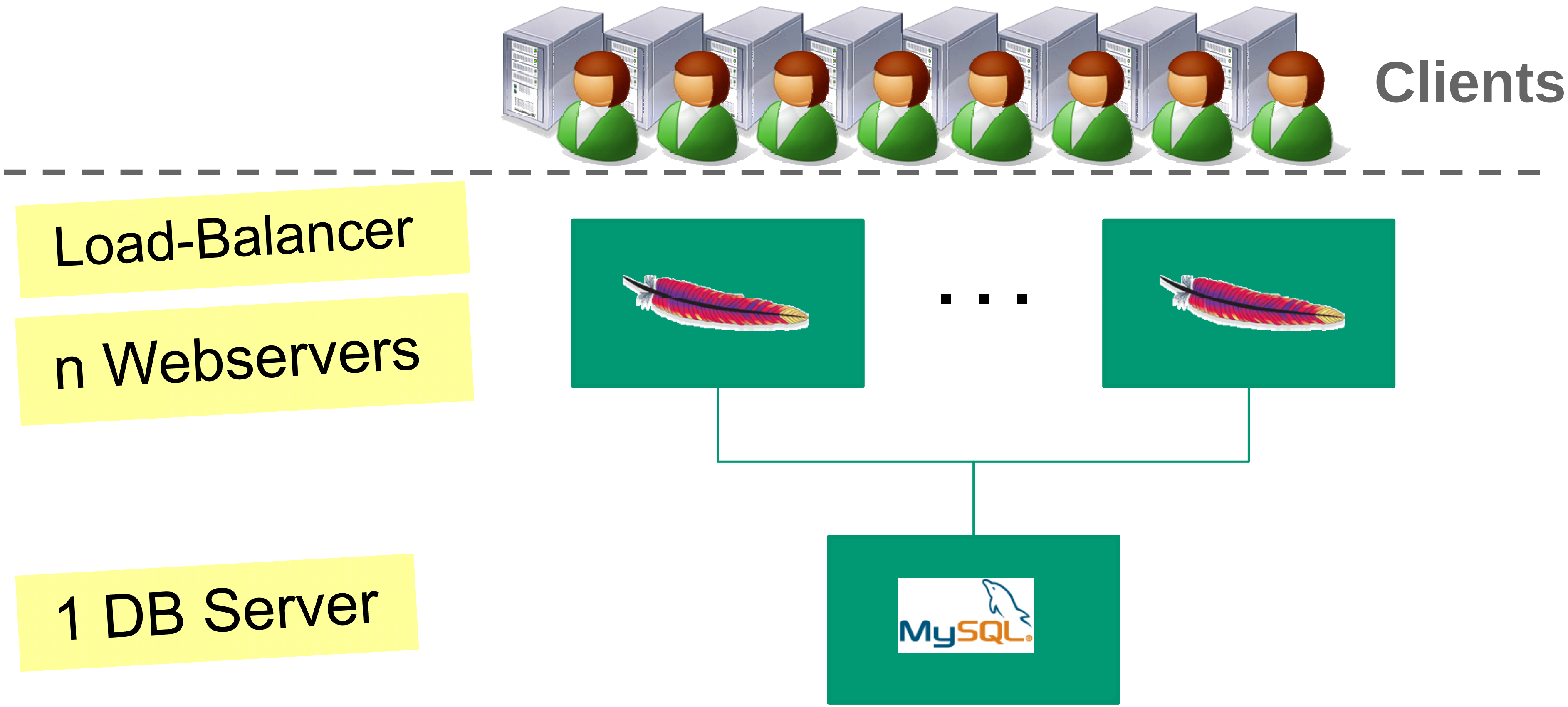
Clients

1 Webserver

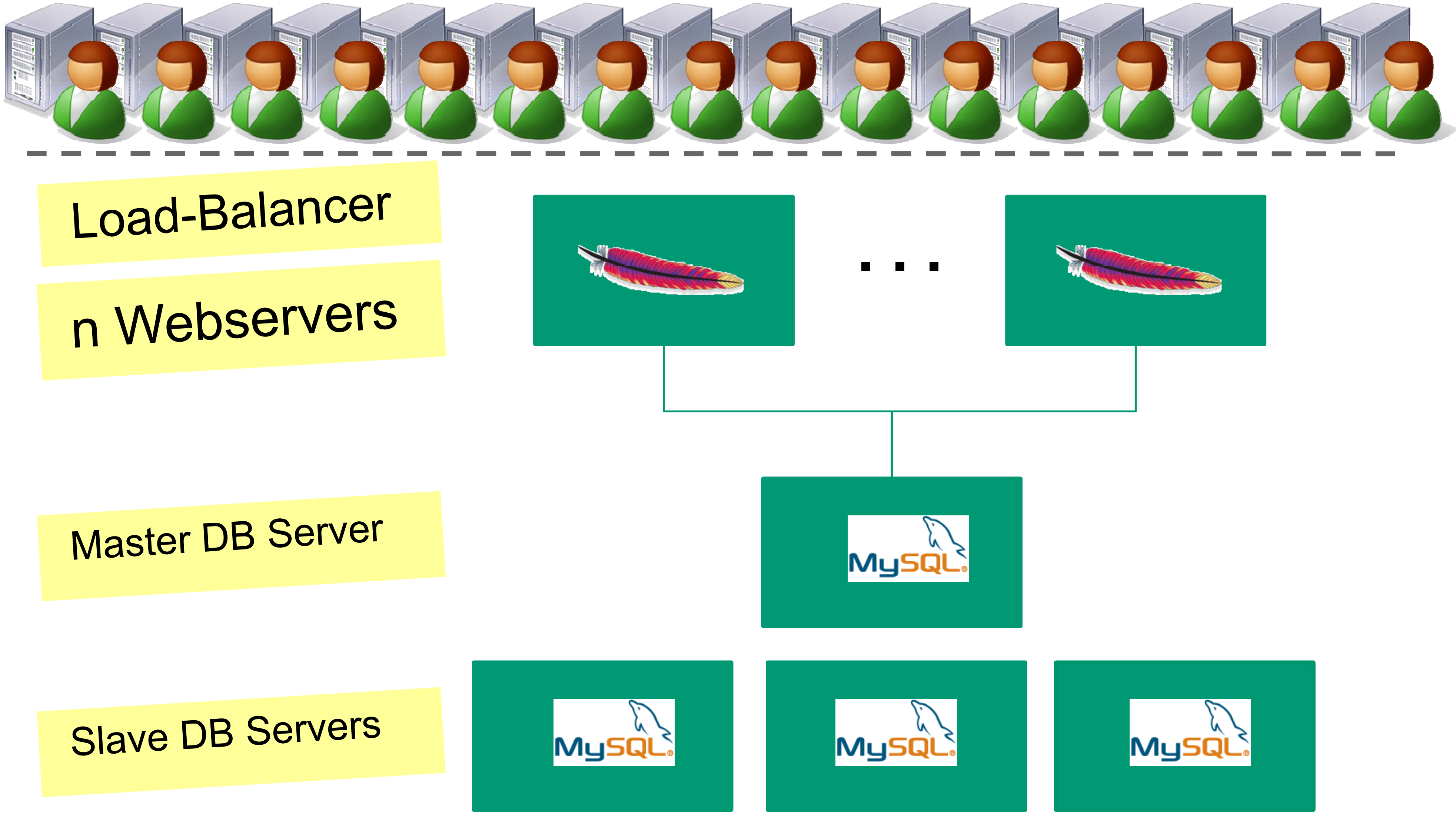
1 DB Server



Traditional Web 2.0 hosting



Traditional Web 2.0 hosting



Now think big.
Really BIG.

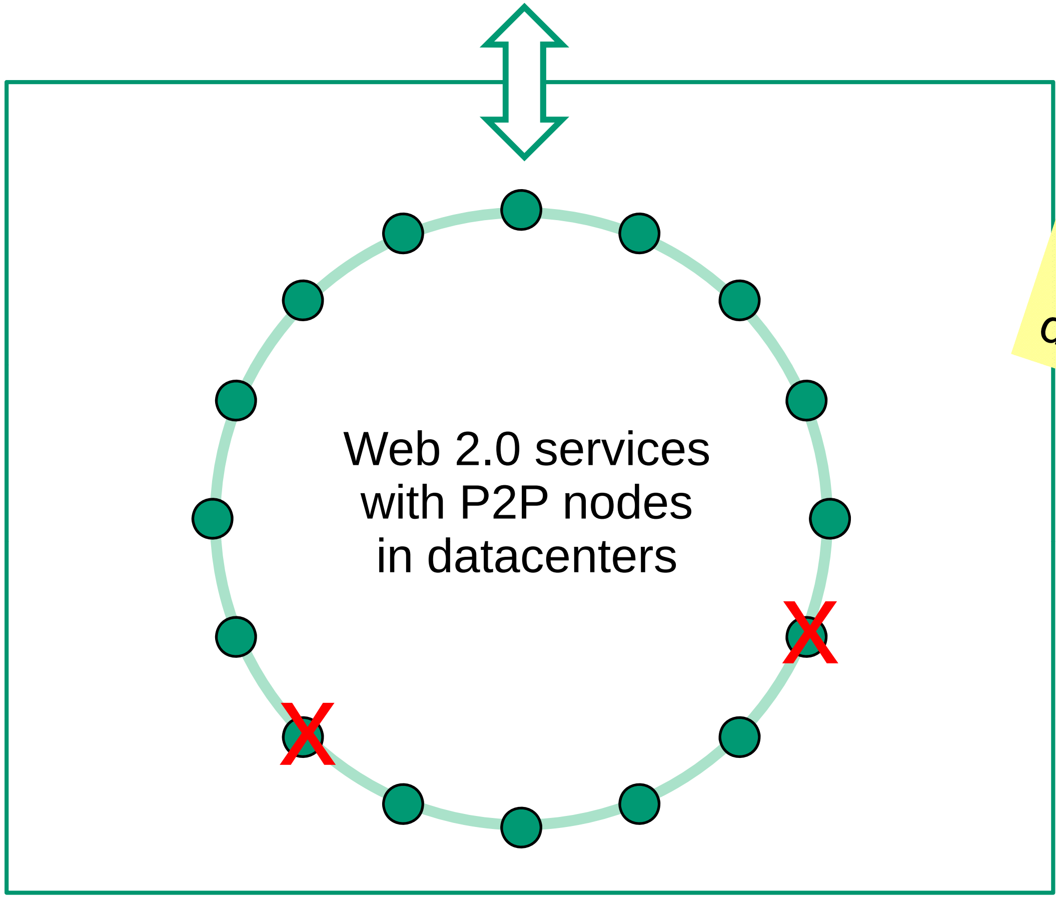
Not how fast our code is today, but:

- Can it “scale out”?
- Can it run in parallel? … distributed?
- Any common resources causing locking?

Asymptotic performance matters!

Our Approach: P2P makes it scalable

“arbitrary“ number of clients



scalable, distributed key/value store with strong data consistency

Our Approach

4 Application Layer

crash recovery model

Key/Value Store (= simple database) — strong data consistency

3 Transaction Layer — implements ACID

2 Replication Layer — improves availability at the cost of consistency

1 crash stop model

P2P Layer — implements
- scalability
- eventual consistency

unreliable, distributed nodes

1

providing a scalable distributed data store:

P2P LAYER

Key/Value Store

- for storing “items” (= “key/value pairs”)
  - synonyms: “key/value store”, “dictionary”, “map”, …

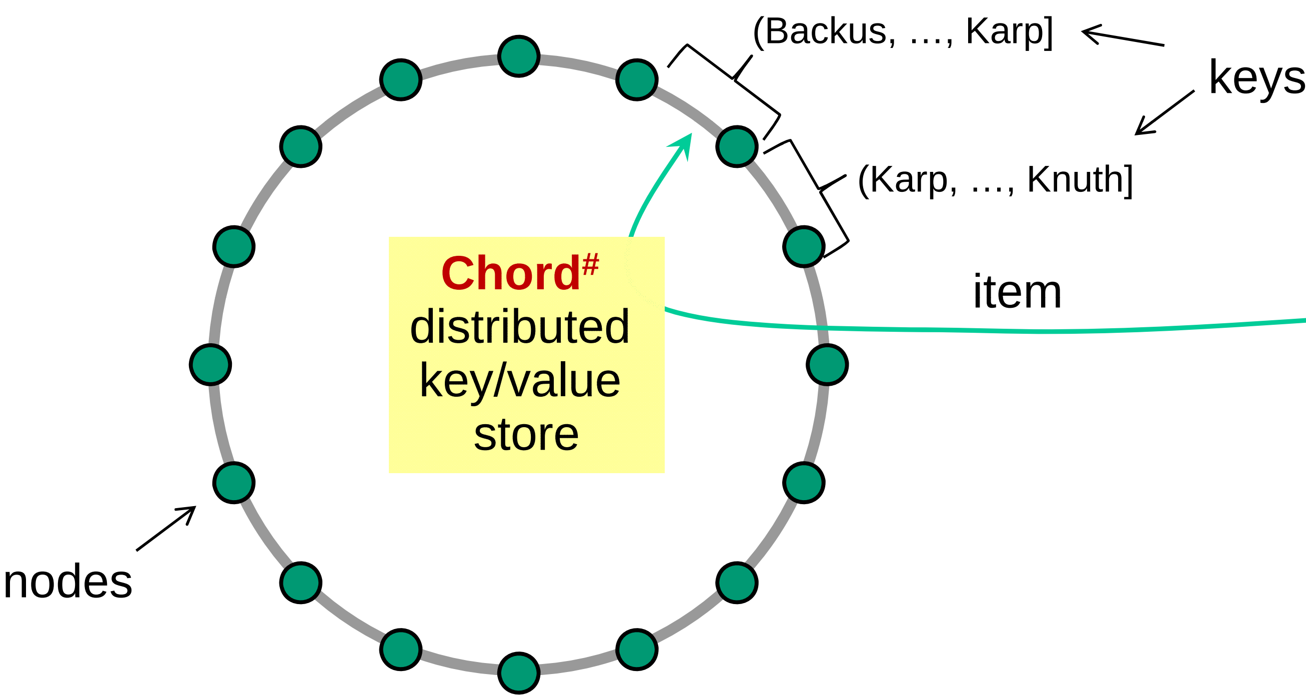
- just 3 ops
  - `insert(key, value)`
  - `delete(key)`
  - `lookup(key)`

Turing Award Winners

Key	Value
Backus	1977
Hoare	1980
Karp	1985
Knuth	1974
Wirth	1984
. . .	. . .

Chord# - Distributed Key/Value Store

- ***key space***: total order on items (strings, numbers, …)
- nodes have a random key as their position in ring
- items are stored on the successor node (clockwise)



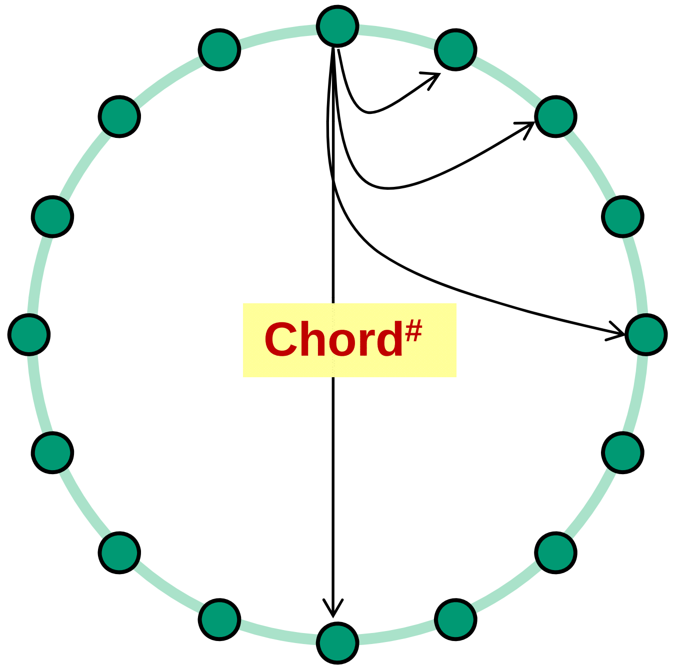
Key	Value
Backus	1977
Hoare	1980
Karp	1985
Knuth	1974
Wirth	1984
. . .	. . .

Routing Table and Data Lookup

Building the routing table

- $\log_2 N$ pointers
- exponentially spaced pointers

$$pointer_i = \begin{cases} successor & : i = 0 \\ pointer_{i-1} \,.\, pointer_{i-1} & : i \neq 0 \end{cases}$$

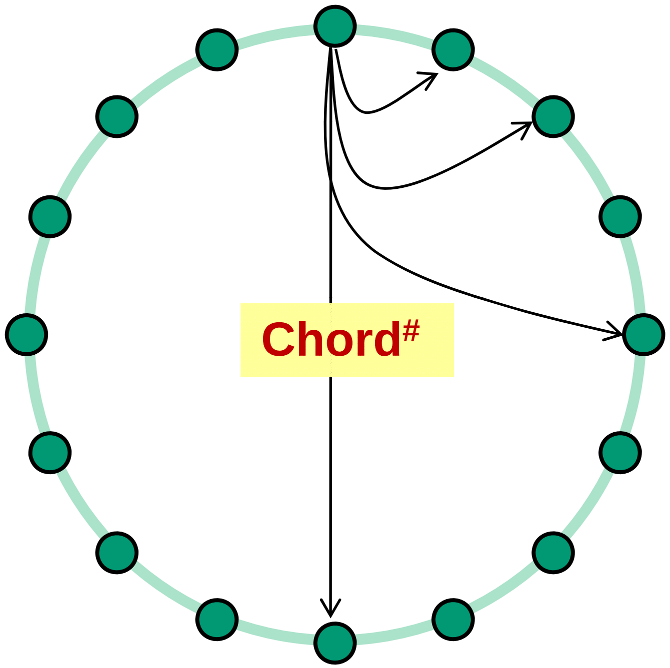


Routing Table and Data Lookup

Building the routing table

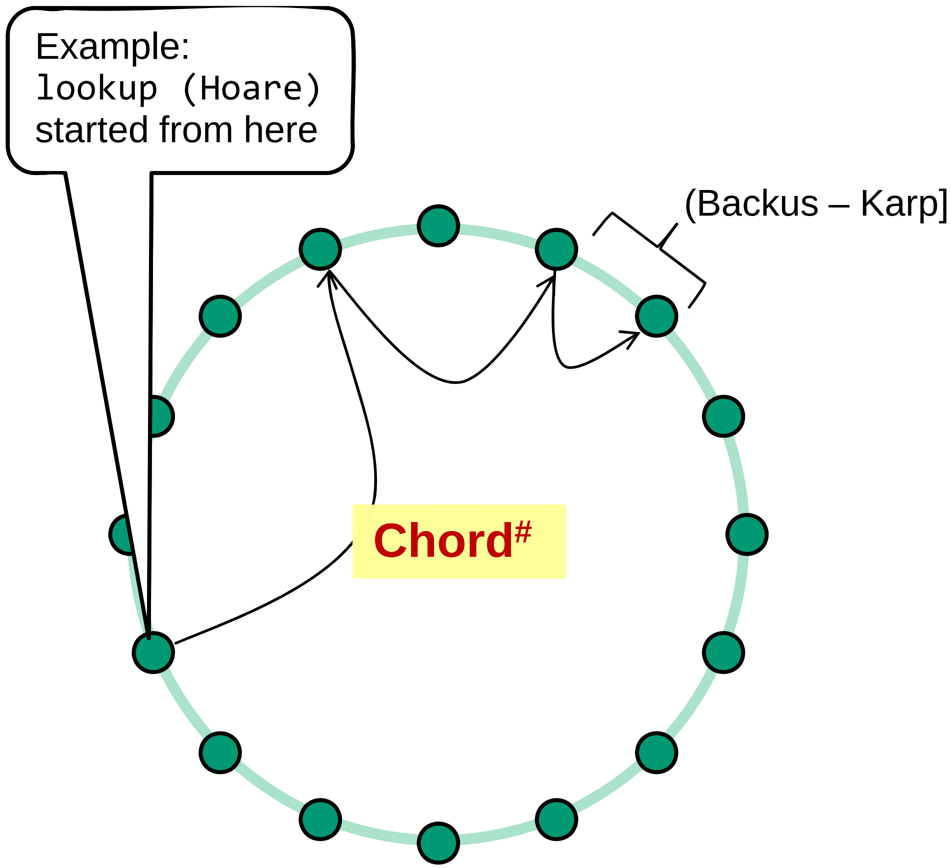
- $\log_2 N$ pointers
- exponentially spaced pointers

$$pointer_i = \begin{cases} successor & : i = 0 \\ pointer_{i-1} \cdot pointer_{i-1} & : i \neq 0 \end{cases}$$



Retrieving items

- $\leq \log_2 N$ hops

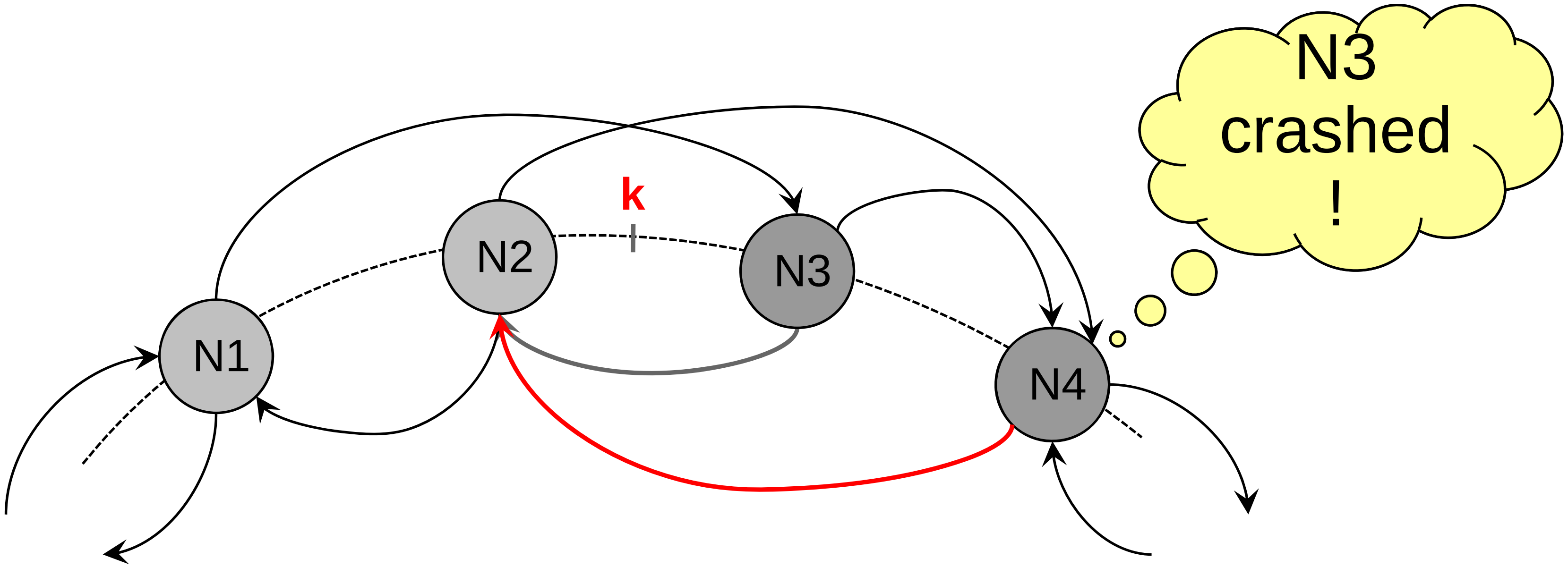


Churn

- Nodes *join*, *leave*, or *crash* at any time
- Need "**failure detector**" to check aliveness of nodes
  - failure detector may be wrong: Node dead? Or just slow network?
- Churn may cause inconsistencies
  - need local repair mechanism

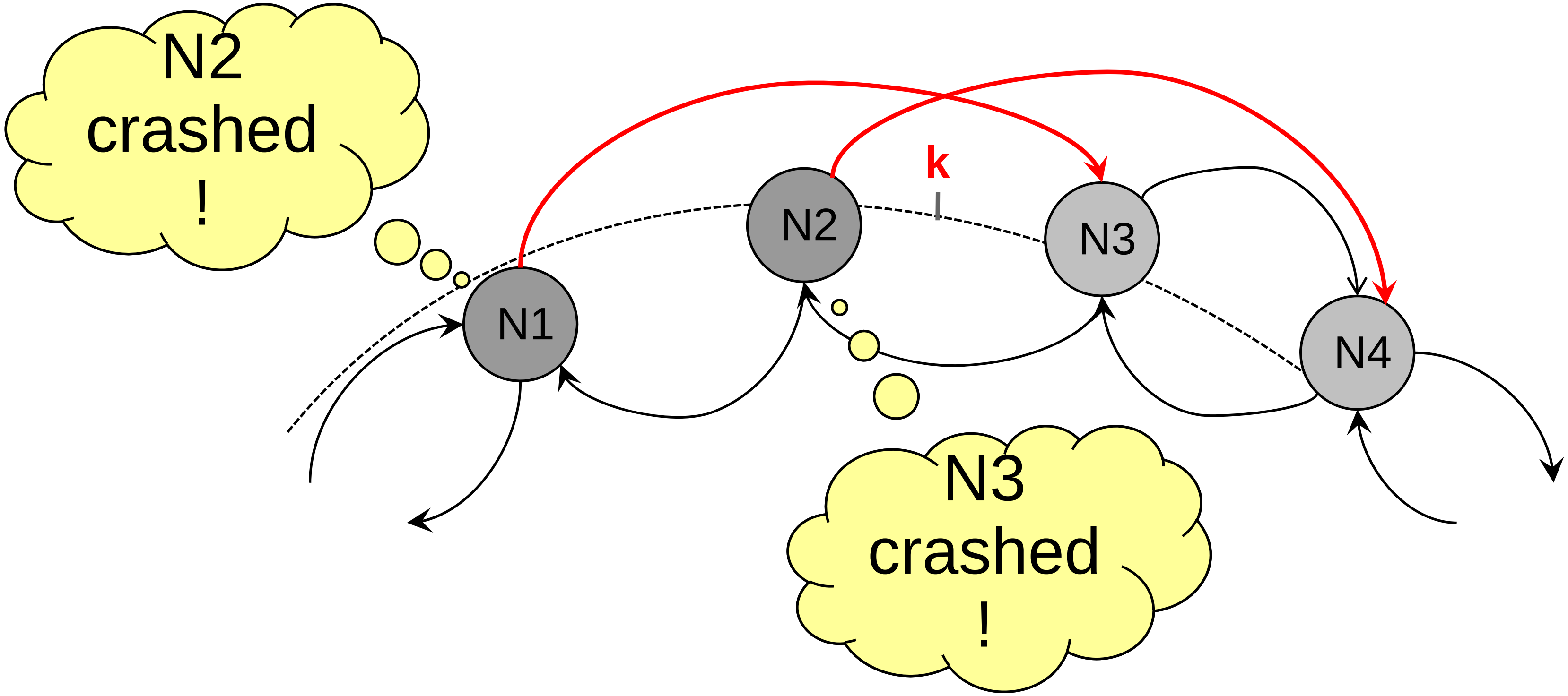
Responsibility Consistency

- **Violated responsibility consistency** caused by imperfect failure detector: Both, N3 ***and*** N4 claim responsibility for item **k**



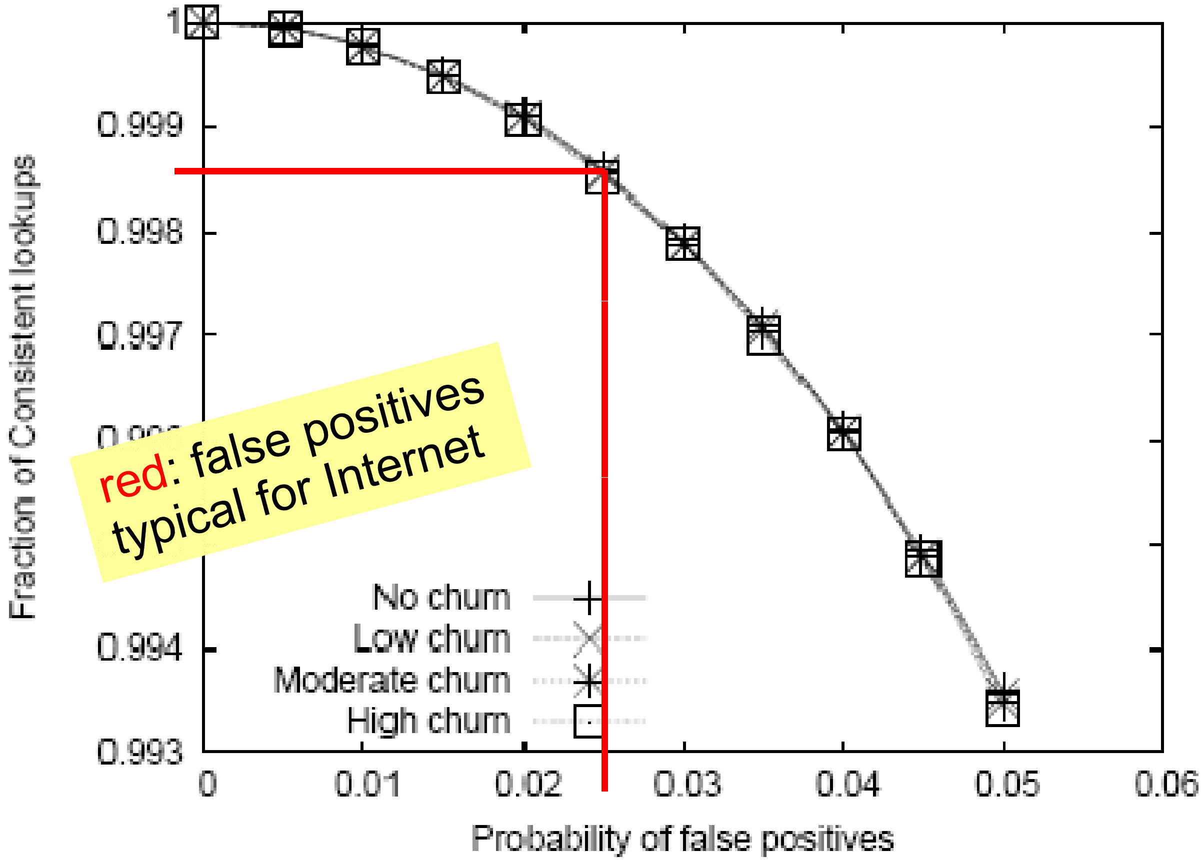
Lookup Consistency

- **Violated lookup consistency** caused by imperfect failure detector: `lookup(k):` at N1 → N3, but at N2 → N4



How often does this occur?

- Simulated nodes with imperfect failure detectors
  (A node detects another alive node as dead probabilistically)



SUMMARY

1

P2P LAYER

- Chord# provides a key/value store
  - scalable
  - efficient: $\log_2 N$ hops
- Quality of failure detector is crucial
- Need **replication** to prevent data loss …

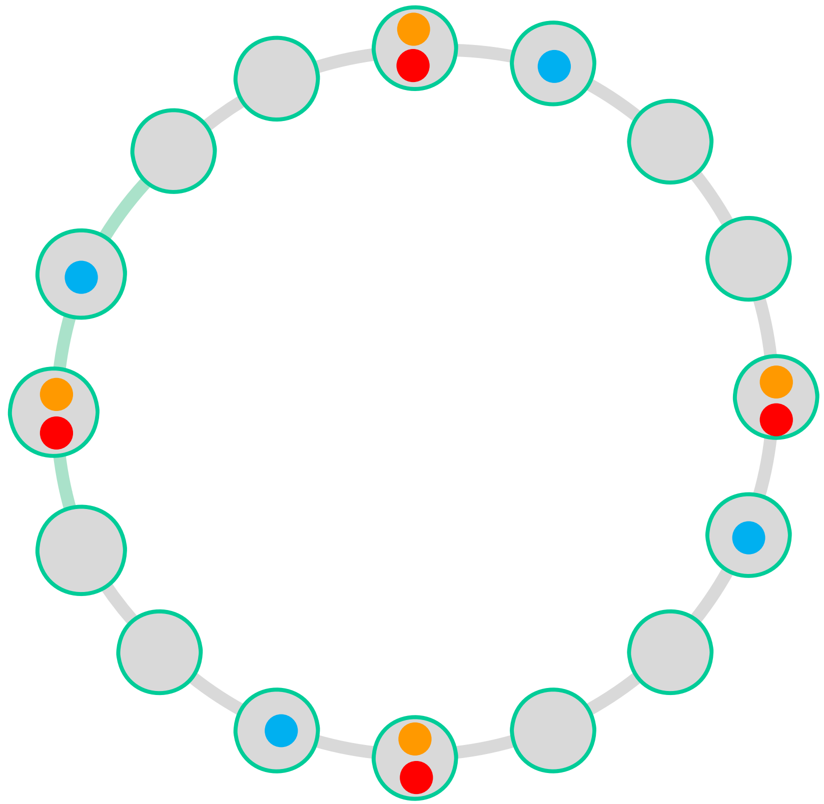
2

improving availability

REPLICATION LAYER

Replication

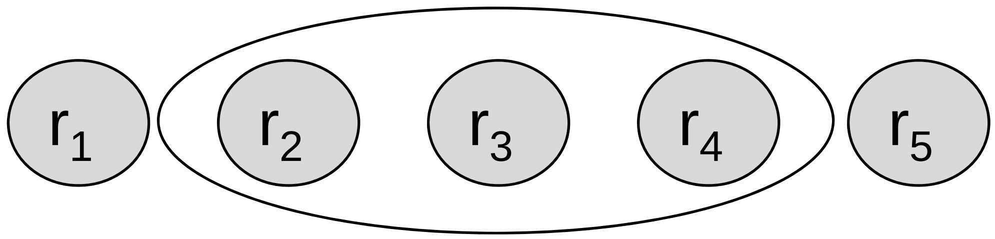
- Many schemes
  - symmetric replication →
  - succ. list replication
  - …



- Must ensure data consistency
  - need quorum-based methods

Quorum based algorithms

- Enforce consistency by operating on majorities



majority

- Comes at the cost of increased latency
  - but latency can be avoided by clever replica distribution in datacenters (**cloud computing**)

SUMMARY

2

REPLICATION LAYER

- availability in face of churn
- quorum algorithms
- But need **transactional** data access …

3

coping with concurrency:

TRANSACTION LAYER

Transaction Layer

- Transactions on P2P are challenging because of …
  - **churn**
    - changing node responsibilities
  - **crash stop** fault model
    - as opposed to crash recovery in traditional DBMS
  - imperfect **failure detector**
    - don’t know whether node crashed or slow network

Strong Data Consistency

- What is it?
  - When a write is finished, all following reads return the new value.

- How to implement?
  - Always read/write ***majority*** $\lfloor f/2 \rfloor + 1$ of f replicas.

    → Latest version is always in the read or write set

  - Must ensure that replication degree is ≤ f

Atomicity

- What is it?
  - Make **all** or **no** changes!
  - Either ‘commit’ or ‘abort’.

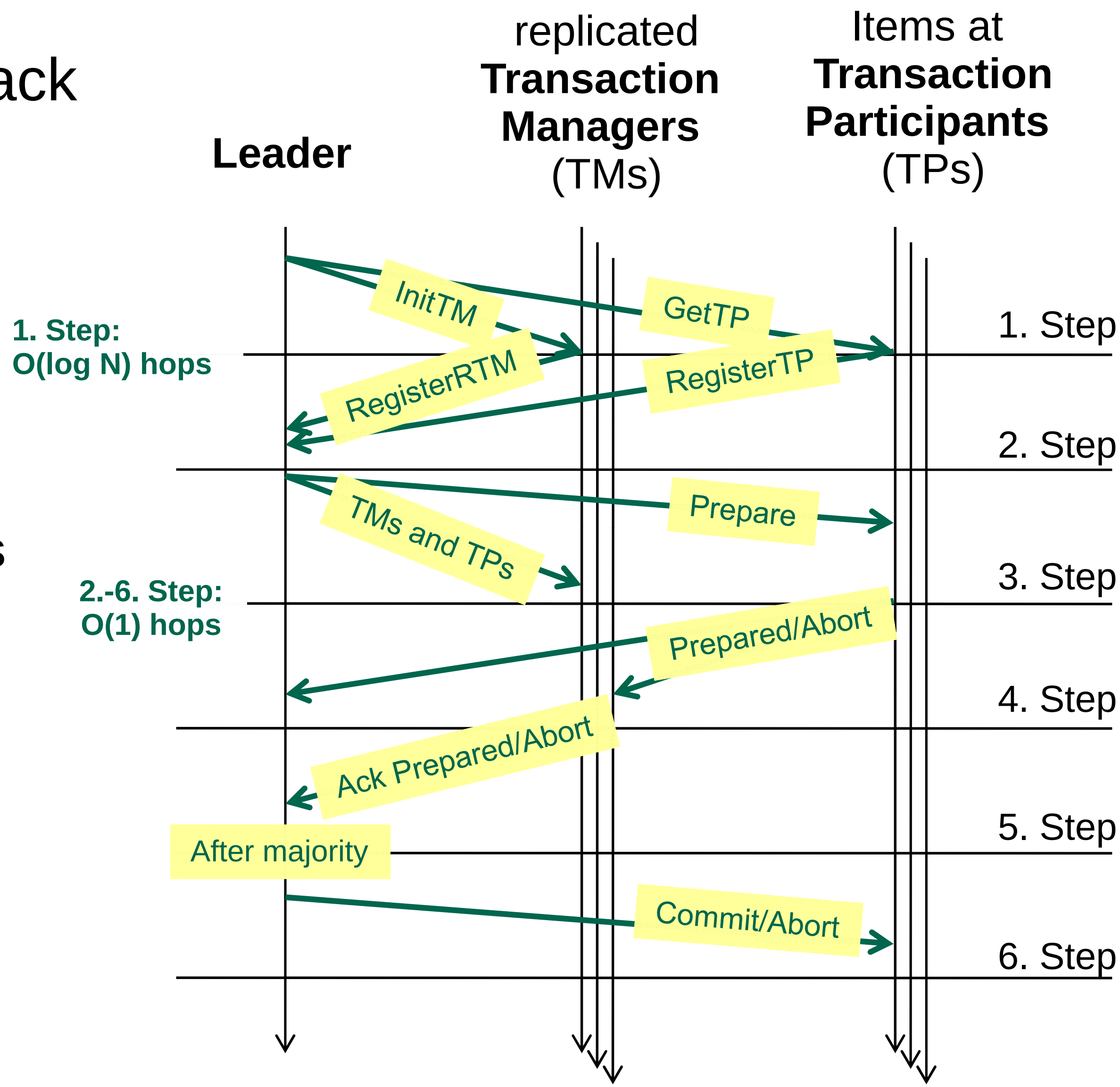
- How to implement?
  - 2PC? Blocks if the transaction manager fails.
  - 3PC? Too much latency.
  - We use a variant of the **Paxos Commit** Protocol
    - non-blocking: Votes of transaction participants are sent to multiple “acceptors”

Adapted Paxos Commit

- Optimistic CC with fallback
- Write
  - 3 rounds
  - non-blocking (fallback)
- Read even faster
  - reads majority of replicas
  - just 1 round
- succeeds when >f/2 nodes alive

Adapted Paxos Commit

- Optimistic CC with fallback
- Write
  - 3 rounds
  - non-blocking (fallback)
- Read even faster
  - reads majority of replicas
  - just 1 round
- succeeds when
  >f/2 nodes alive



Transactions have two purposes:

Consistency of replicas & consistency across items

User Request

BOT

- debit (a, 100);

- deposit (b, 100);

EOT

Operation on replicas

BOT

- debit (a1, 100);
- debit (a2, 100);
- debit (a3, 100);

- deposit (b1, 100);
- deposit (b2, 100);
- deposit (b3, 100);

EOT

SUMMARY

3

TRANSACTION LAYER

- Consistent update of items and replicas
- Mitigates some of the overlay oddities
  - node failures
  - asynchronicity

4

demonstrator application:

WIKIPEDIA

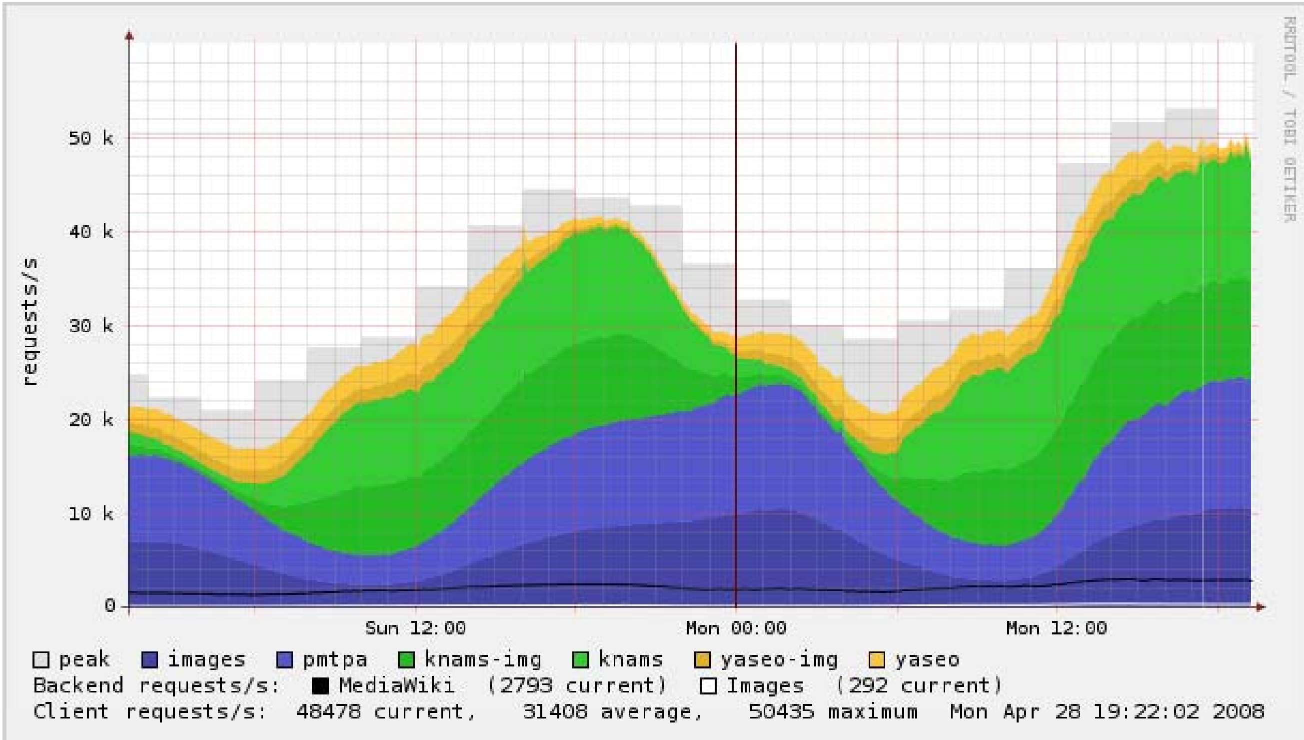
Wikipedia

Top 10 Web sites

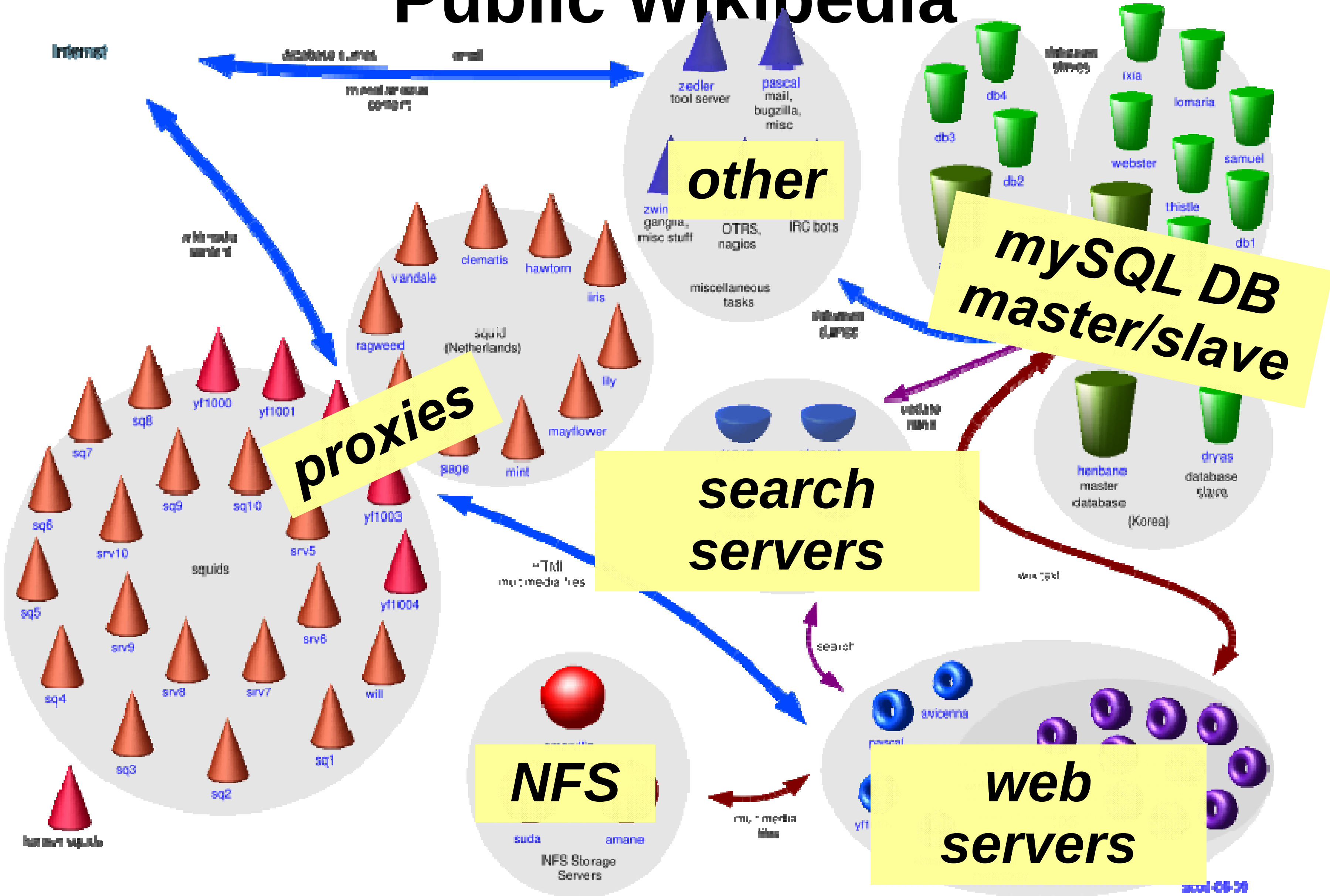
1. Yahoo!
2. Google
3. YouTube
4. Windows Live
5. MSN
6. Myspace
7. **Wikipedia**
8. Facebook
9. Blogger.com
10. Yahoo!カテゴリ

50.000 requests/sec

- 95% are answered by squid proxies
- **only 2,000 req./sec hit the backend**



Public Wikipedia



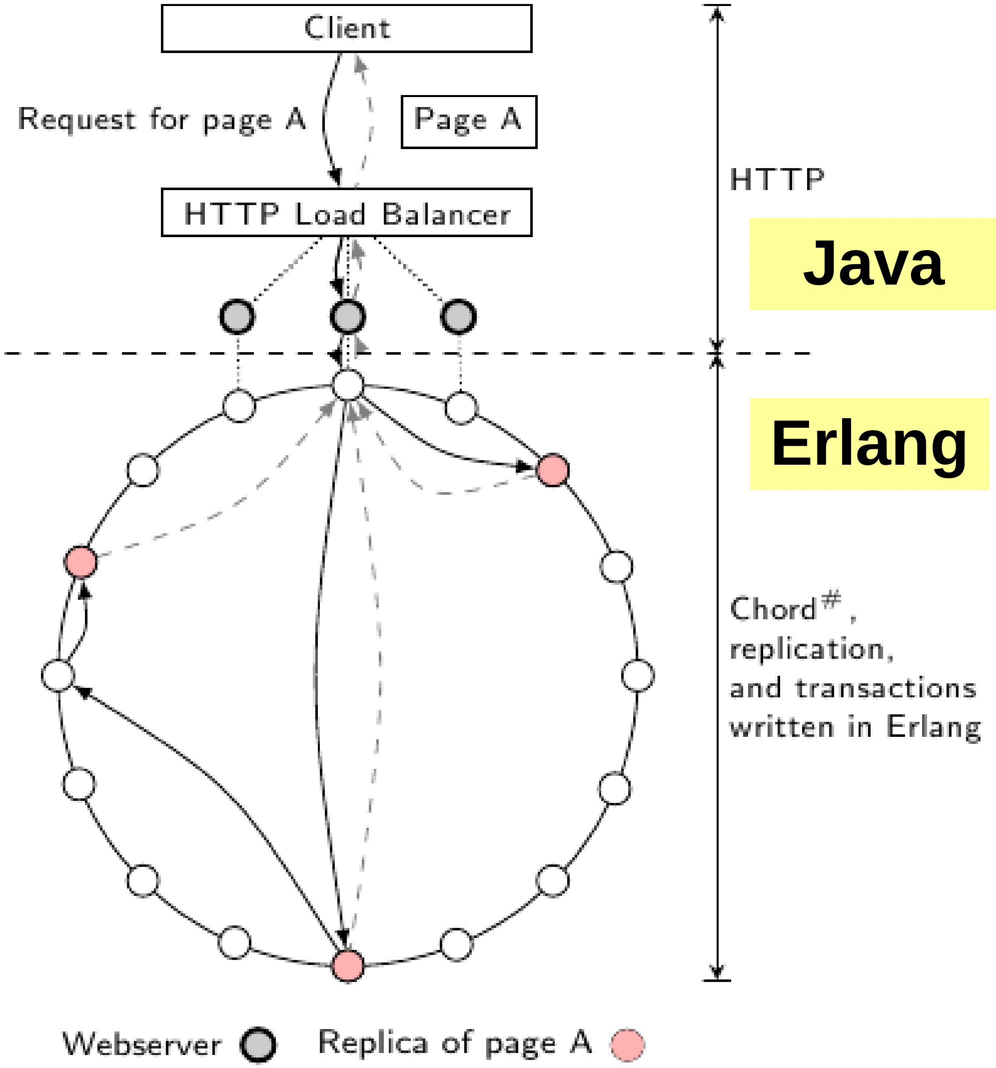
Our Wikipedia

Renderer

- Java
  - Tomcat, Plog4u
- Jinterface
  - Interface to Erlang

Key/Value Store

- Chord# + Replication + Transactions



Mapping Wikipedia to Key/Value Store

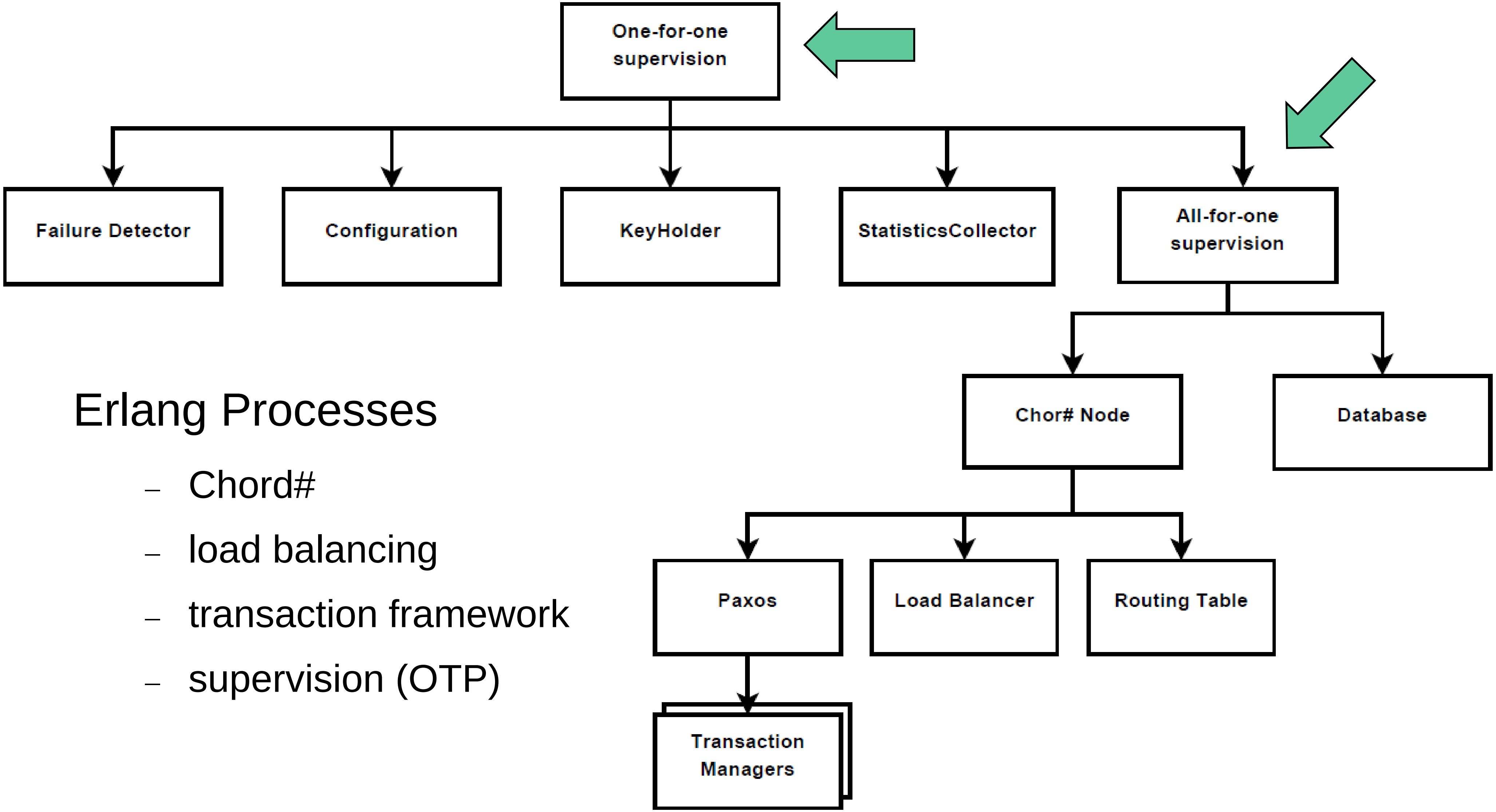
- Mapping

	key	**value**
page content	title	list of Wikitext for all versions
backlinks	title	list of titles
categories	category name	list of titles

- For each insert or modify we must
  - update backlinks
  - update category page(s)

write transaction

Erlang Processes



Erlang Processes

- Chord#
- load balancing
- transaction framework
- supervision (OTP)

Erlang Processes (per node)

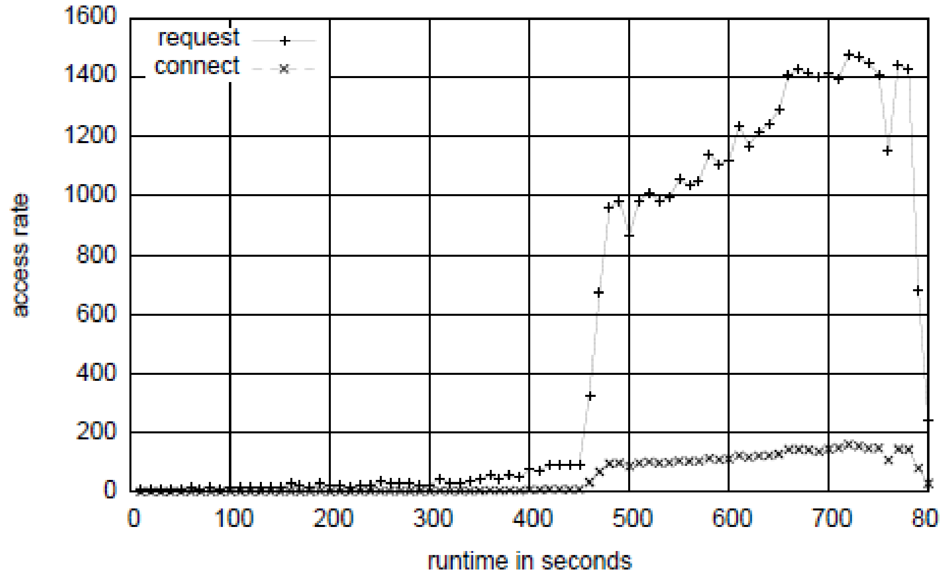
- **Failure Detector** supervises Chord# nodes and sends crash messages when a failure is detected.
- **Configuration** provides access to the configuration file and maintains parameter changes made at runtime.
- **Key Holder** stores the identifier of the node in the overlay.
- **Statistics Collector** collects statistics information and forwards them to statistic servers.
- **Chord# Node** performs the main functionality of the node, e.g. successor list and routing table.
- **Database** stores key/value pairs in each node.

Accessing Erlang Transactions from Java via Jinterface

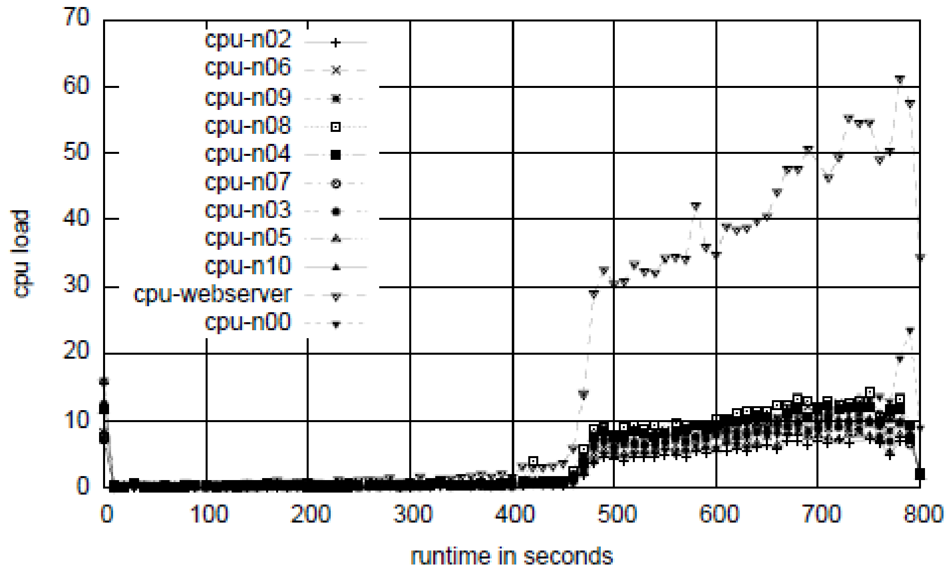
```
void updatePage(string title, int oldVersion, string newText)
{
   Transaction t = new Transaction();    //new transaction
   Page p = t.read(title);               // read old version
   if (p.currentVersion != oldVersion)   // concurrent update?
       t.abort();
   else {
       t.write(p.add(newText));          // write new text
       //update categories
       foreach(Category c in p)
           t.write(t.read(c.name).add(title));
       t.commit();
   }
}
```

Performance on Linux Cluster

test results with load generator



throughput with increasing access rate over time



CPU load with increasing access rate over time

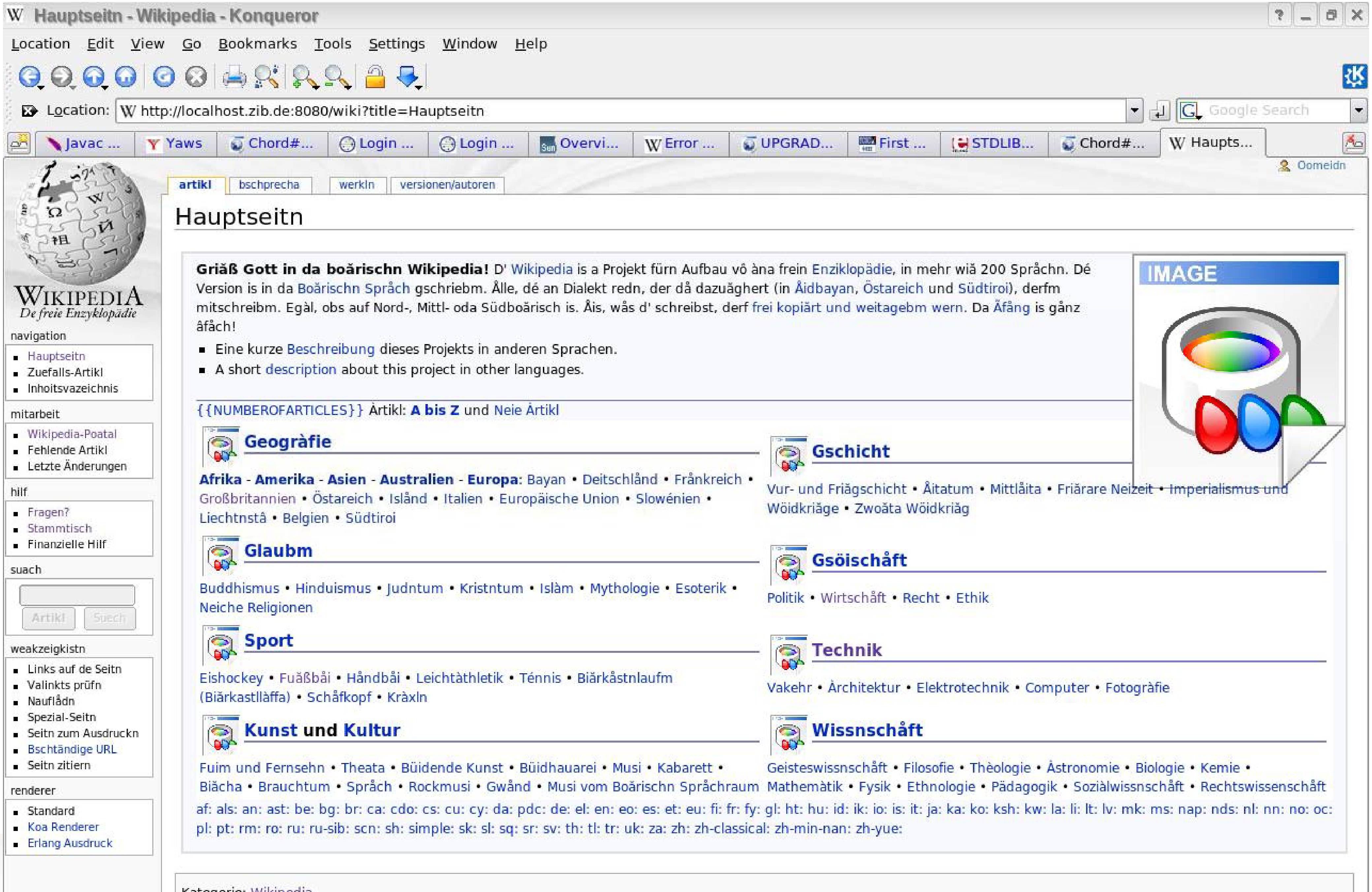
1500 trans./sec on 10 CPUs

2500 trans./sec on 16 CPUs (64 cores) and 128 DHT nodes

Implementation

- 11,000 lines of Erlang code
  - 2,700 for transactions
  - 1,300 for Wikipedia
  - 7,000 for Chord# and infrastructure
- Distributed Erlang
  - currently has weak security and limited scalability

    $\Rightarrow$ we implemented own transport layer on top of TCP
- Java for rendering and user interface


Hauptseitn - Wikipedia - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://localhost.zib.de:8080/wiki?title=Hauptseitn
Google Search
Javac ... Yaws Chord#... Login ... Login ... Overvi... Error ... UPGRAD... First ... STDLIB... Chord#... Haupts...
Oomeidn
artikl bschprecha werkln versionen/autoren
Hauptseitn
WIKIPEDIA
De freie Enzyklopädie
navigation
Hauptseitn
Zuefalls-Artikl
Inhoitsvazeichnis
mitarbeit
Wikipedia-Poatal
Fehlende Artikl
Letzte Änderungen
hilf
Fragen?
Stammtisch
Finanzielle Hilf
suach
Artikl Suech
weakzeigkistn
Links auf de Seitn
Valinkts prüfn
Nauflådn
Spezial-Seitn
Seitn zum Ausdruckn
Bschtändige URL
Seitn zitiern
renderer
Standard
Koa Renderer
Erlang Ausdruck
Griäß Gott in da boărischn Wikipedia! D' Wikipedia is a Projekt fürn Aufbau vô ana frein Enziklopädie, in mehr wiä 200 Språchn. Dé Version is in da Boărischn Språch gschriebm. Älle, dé an Dialekt redn, der då dazuăghert (in Äidbayan, Östareich und Südtiroi), derfm mitschreibm. Egàl, obs auf Nord-, Mittl- oda Südboărisch is. Åis, wås d' schreibst, derf frei kopiärt und weitagebm wern. Da Åfång is gånz åfåch!
Eine kurze Beschreibung dieses Projekts in anderen Sprachen.
A short description about this project in other languages.
IMAGE
{{NUMBEROFARTICLES}} Artikl: A bis Z und Neie Artikl
Geogràfie
Afrika - Amerika - Asien - Australien - Europa: Bayan • Deitschlånd • Frånkreich • Großbritannien • Östareich • Islånd • Italien • Europäische Union • Slowénien • Liechtenstå • Belgien • Südtiroi
Gschicht
Vur- und Friägschicht • Åitatum • Mittlåita • Friărare Neizeit • Imperialismus und Wöidkriäge • Zwoăta Wöidkriäg
Glaubm
Buddhismus • Hinduismus • Judntum • Kristntum • Islàm • Mythologie • Esoterik • Neiche Religionen
Gsöischåft
Politik • Wirtschåft • Recht • Ethik
Sport
Eishockey • Fuaßbåi • Håndbåi • Leichtàthletik • Ténnis • Biärkåstnlaufm (Biärkastllåffa) • Schåfkopf • Kràxln
Technik
Vakehr • Àrchitektur • Elektrotechnik • Computer • Fotogràfie
Kunst und Kultur
Fuim und Fernsehn • Theata • Büidende Kunst • Büidhauarei • Musi • Kabarett • Biächa • Brauchtum • Språch • Rockmusi • Gwånd • Musi vom Boărischn Språchraum
Wissnschåft
Geisteswissnschåft • Filosofie • Thèologie • Åstronomie • Biologie • Kemie • Mathemàtik • Fysik • Ethnologie • Pädagogik • Soziàlwissnschåft • Rechtswissenschåft
af: als: an: ast: be: bg: br: ca: cdo: cs: cu: cy: da: pdc: de: el: en: eo: es: et: eu: fi: fr: fy: gl: ht: hu: id: ik: io: is: it: ja: ka: ko: ksh: kw: la: li: lt: lv: mk: ms: nap: nds: nl: nn: no: oc: pl: pt: rm: ro: ru: ru-sib: scn: sh: simple: sk: sl: sq: sr: sv: th: tl: tr: uk: za: zh: zh-classical: zh-min-nan: zh-yue:
Kategorie: Wikipedia

5 SUMMARY

Summary

- P2P as new paradigm for Web 2.0 hosting
  - we support consistent, distributed write operations.

- Numerous applications:
  - Internet databases, transactional online-services, …

All Things Distributed

Werner Vogels' weblog on building scalable and robust distributed systems.

Eventually Consistent

By Werner Vogels on December 19, 2007 10:03 PM | Permalink | Comments (16) | TrackBacks (6)

Recently there has been a lot of discussion about the concept of *eventual consistency* in the context of data replication. In this positing I would like to try to collect some of the principles and abstractions related to large scale data replication and the trade-offs between high-availability and data consistency. I consider this work-in-progress as I don't expect to get every definition crisp the first time.

There are two ways of looking at consistency. One is from the developer / client point of view; how they observe data updates. The second way is from the server side; how updates flow through the system and what guarantees systems can give with respect to updates.

Historical

In an ideal world there would only be one consistency model; when an update is made all observers will see that update. The first time this surfaced as difficult to achieve was in the database systems in the late seventies. The best "period piece" on this topic is by Bruce Lindsay, et al, "Notes on Distributed Databases", Research Report RJ2571(33471), IBM Research, July 1979. The fundamental principles for database replication are laid

Contact Info

Werner Vogels
CTO - Amazon.com

werner@allthingsdistributed.com

Tradeoff: High availability vs. data consistency

Team

- Thorsten Schütt
- Florian Schintke
- Monika Moser
- Stefan Plantikow
- Alexander Reinefeld
- Nico Kruber
- Christian von Prollius
- Seif Haridi (SICS)
- Ali Ghodsi (SICS)
- Tallat Shafaat (SICS)

Publications

Chord#

T. Schütt, F. Schintke, A. Reinefeld.
A Structured Overlay for Multi-dimensional Range Queries. Euro-Par, August 2007.

T. Schütt, F. Schintke, A. Reinefeld.
Structured Overlay without Consistent Hashing: Empirical Results. GP2PC, May 2006.

Transactions

M. Moser, S. Haridi.
Atomic Commitment in Transactional DHTs.
1st CoreGRID Symposium, August 2007.

T. Shafaat, M. Moser, A. Ghodsi , S. Haridi,
T. Schütt, A. Reinefeld. **Key-Based Consistency and Availability in Structured Overlay Networks.**
Infoscale, June 2008.

Wiki

S. Plantikow, A. Reinefeld, F. Schintke.
Transactions for Distributed Wikis on Structured Overlays.
DSOM, October 2007.

Talks / Demos

IEEE Scale Challenge, May 2008
1st price (live demo)