Databases : Lectures 11 and 12: Beyond ACID/Relational databases

Timothy G. Griffin

Lent Term 2013

- Rise of Web and cluster-based computing
- “NoSQL” Movement
- Relationships vs. Aggregates
- Key-value store
- XML or JSON as a data exchange language
- Not all applications require ACID
- CAP = Consistency, Availability, and Partition tolerance
- The CAP theorem (pick any two?)
- Eventual consistency
- Can a database really be “schemaless”?

Eric Brewer’s PODC Keynote (July 2000)

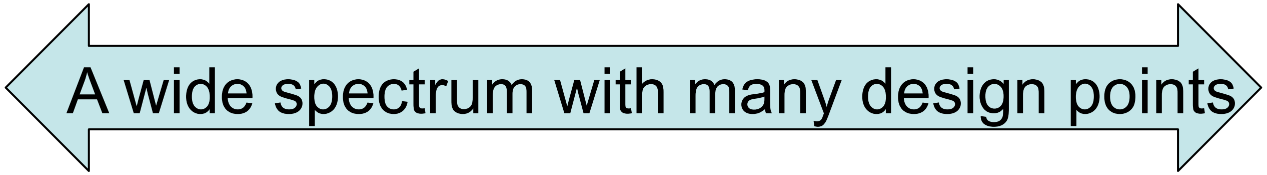
ACID vs. BASE (Basically Available, Soft-state, Eventually consistent)

ACID

- **Strong consistency**
- **Isolation**
- **Focus on “commit”**
- **Nested transactions**
- **Availability?**
- **Conservative (pessimistic)**
- **Difficult evolution (e.g. schema)**

BASE

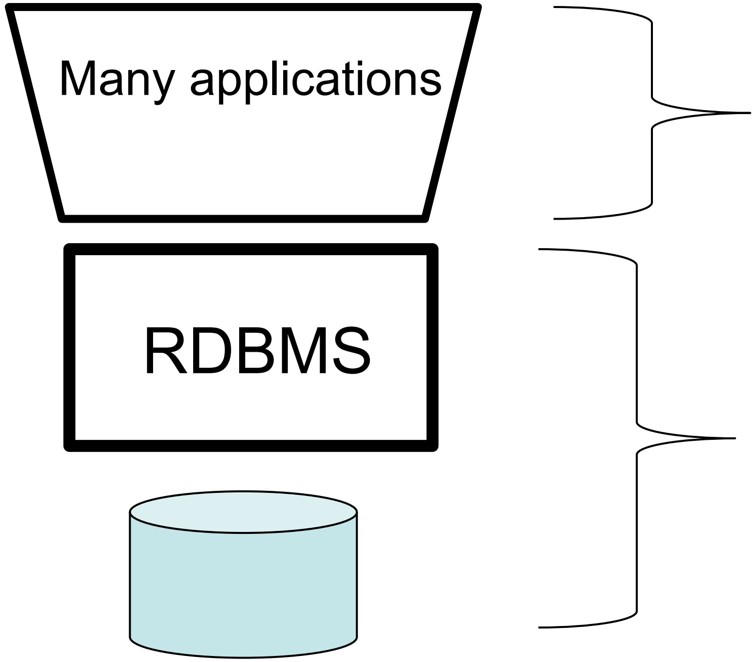
Weak consistency
Availability first
Best effort
Approximate answers OK
Aggressive (optimistic)
Simpler!
Faster
Easier evolution



A wide spectrum with many design points

“Real internet systems are a careful *mixture* of ACID and BASE subsystems”

Relation DB design encourages thinking about data independent of particular applications

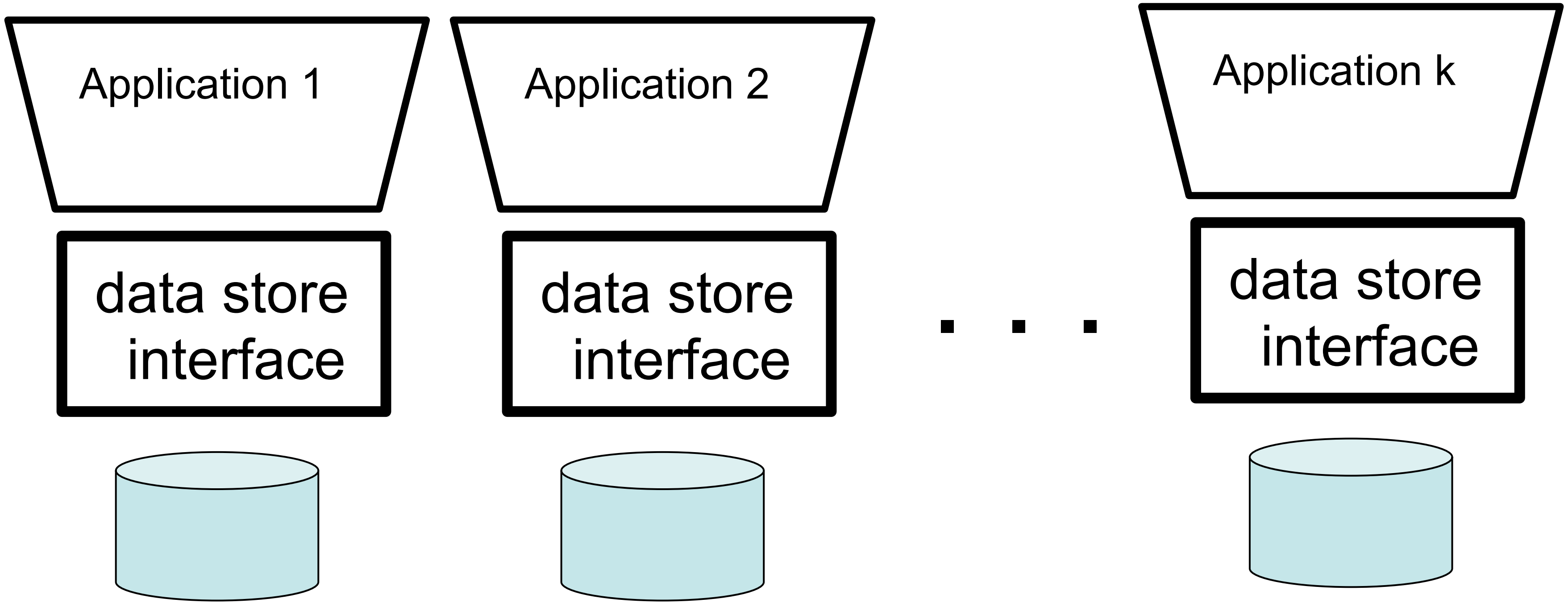


Free to evolve without changing schema

Data integrity and ACID abstractions handled here

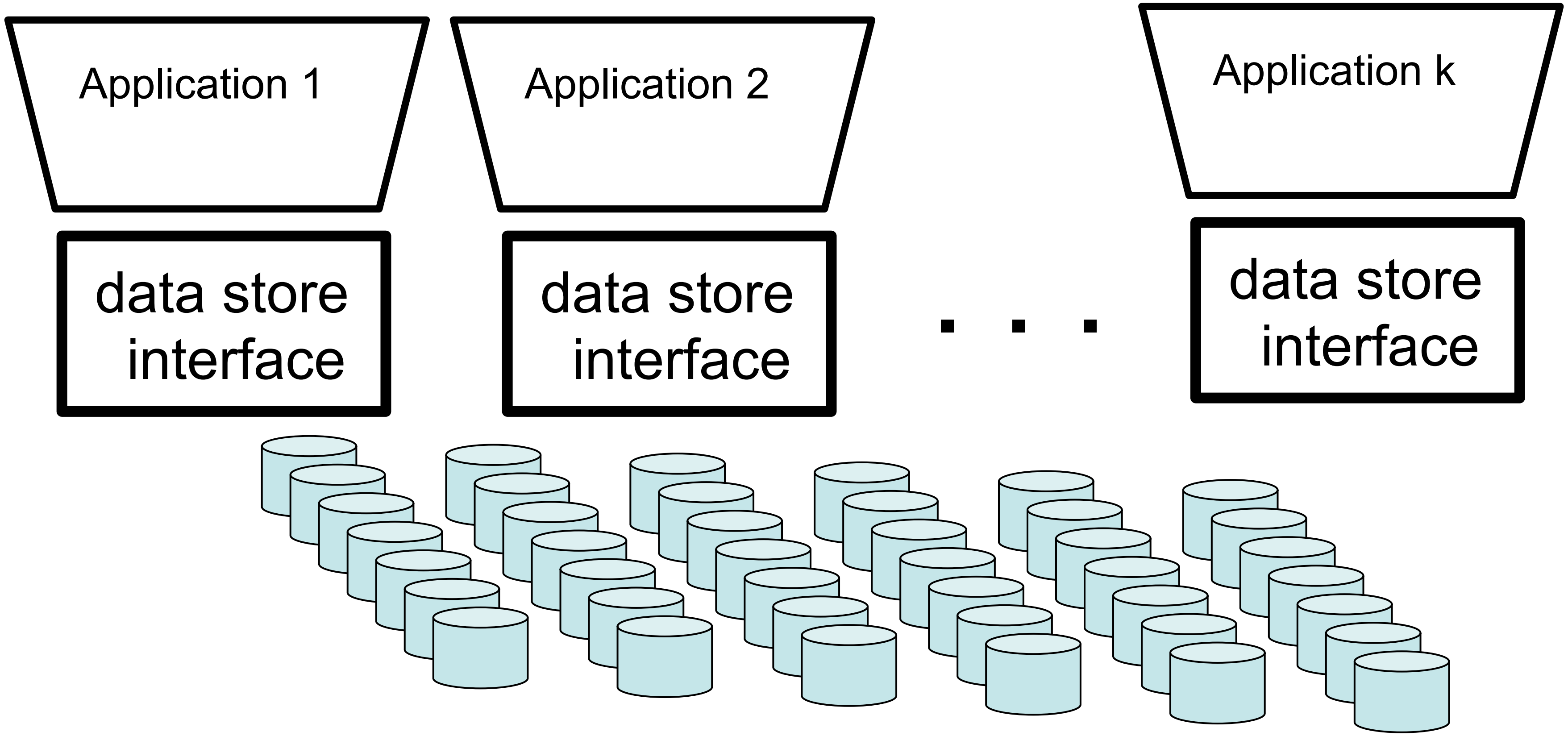
Applications-oriented data stores

Disadvantage: Data integrity and consistency managed “in the application code”



Applications-oriented data stores

Advantage: works well on computing clusters

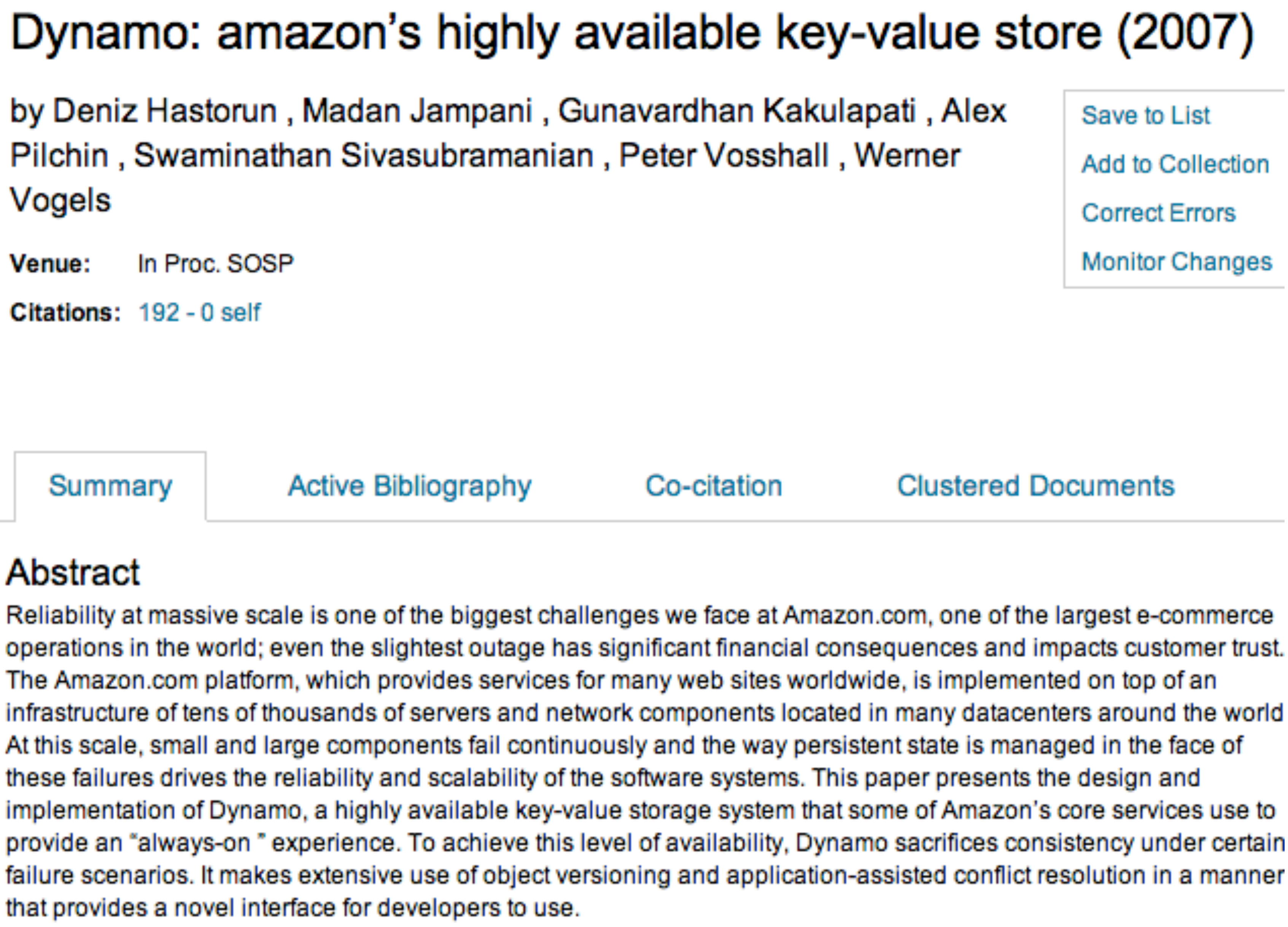


Fallacies of Distributed Computing (Peter Deutsch)

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

https://blogs.oracle.com/jag/resource/Fallacies.html



Dynamo: amazon’s highly available key-value store (2007)

by Deniz Hastorun , Madan Jampani , Gunavardhan Kakulapati , Alex Pilchin , Swaminathan Sivasubramanian , Peter Vosshall , Werner Vogels

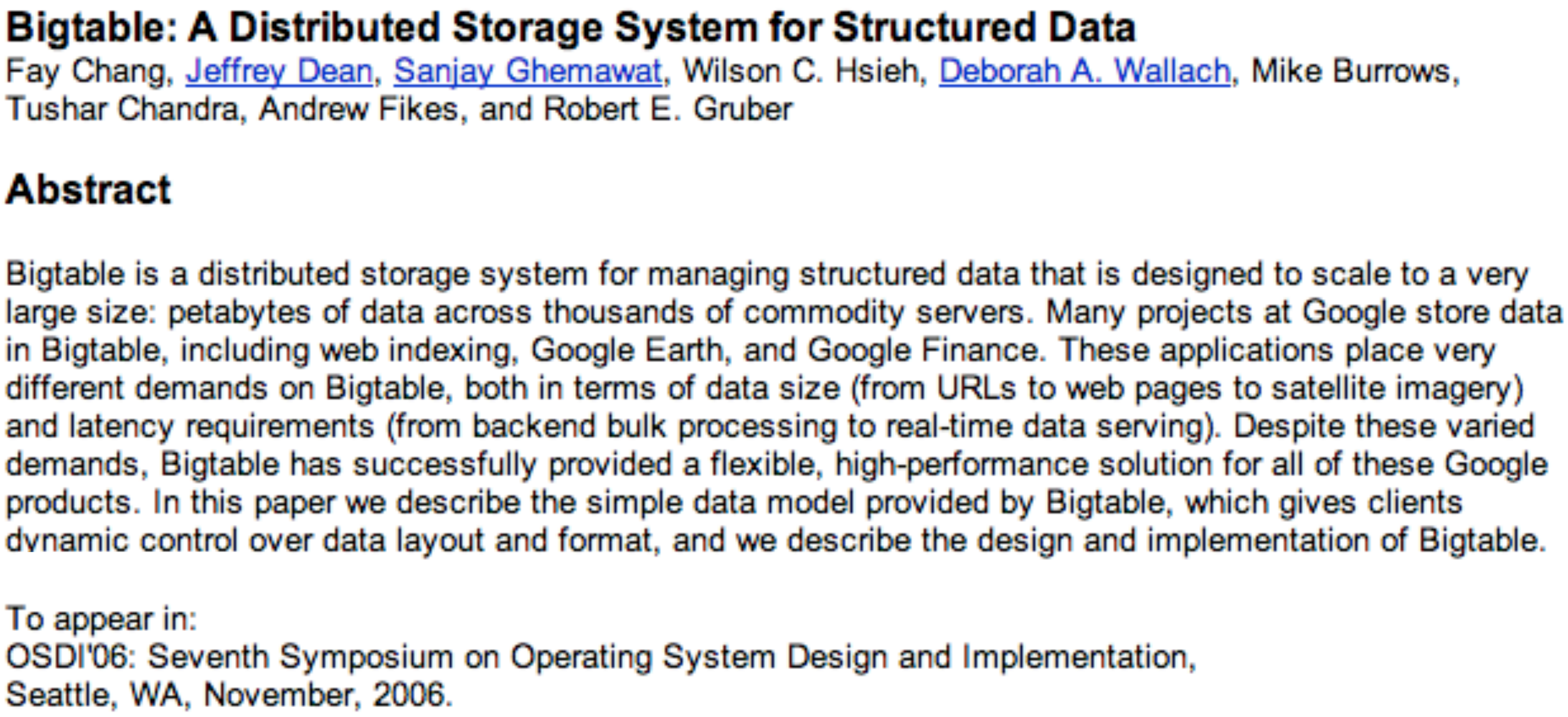
Venue: In Proc. SOSP

Citations: 192 - 0 self

Abstract

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems. This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon’s core services use to provide an “always-on ” experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.6956



Bigtable: A Distributed Storage System for Structured Data

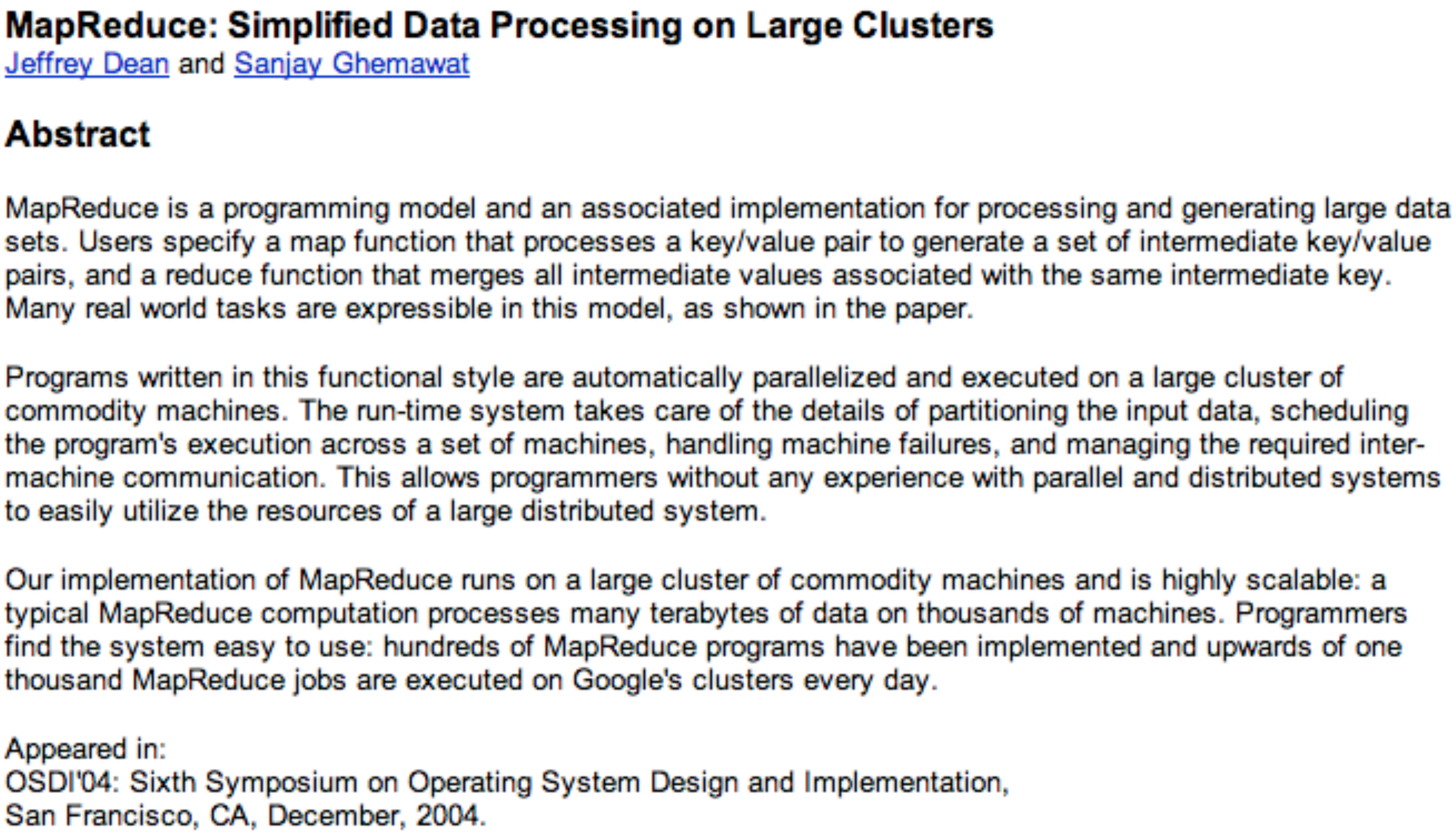
Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dvnamic control over data layout and format, and we describe the design and implementation of Bigtable.

To appear in:
OSDI'06: Seventh Symposium on Operating System Design and Implementation,
Seattle, WA, November, 2006.

http://research.google.com/archive/bigtable.html



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

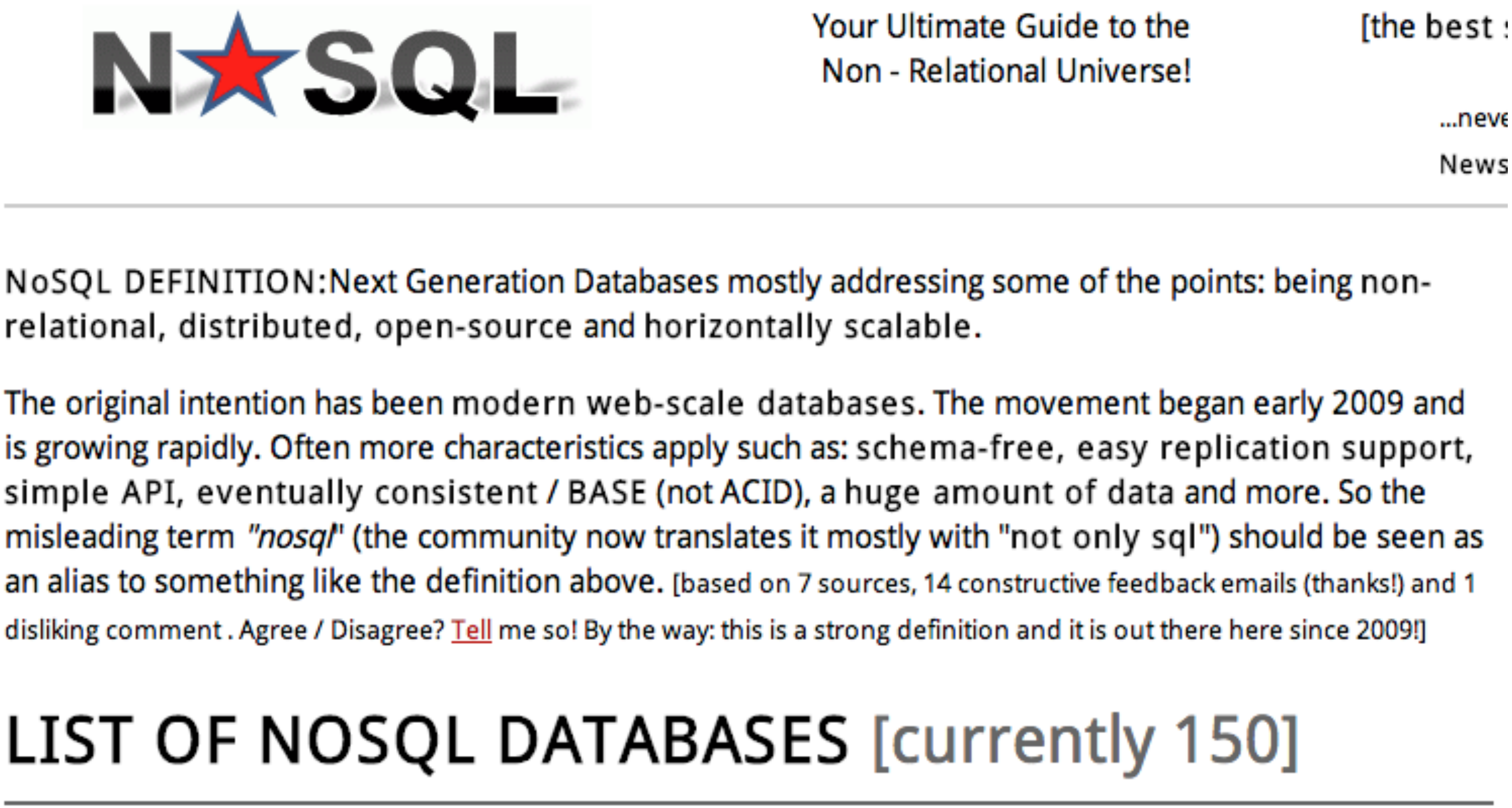
Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

Appeared in:
OSDI'04: Sixth Symposium on Operating System Design and Implementation,
San Francisco, CA, December, 2004.

http://research.google.com/archive/mapreduce.html

http://nosql-database.org/



N★SQL

Your Ultimate Guide to the
Non - Relational Universe!

[the best

...neve

News

NoSQL DEFINITION:Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.

The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more. So the misleading term *"nosql"* (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above. [based on 7 sources, 14 constructive feedback emails (thanks!) and 1 disliking comment . Agree / Disagree? Tell me so! By the way: this is a strong definition and it is out there here since 2009!]

LIST OF NOSQL DATABASES [currently 150]

Application-specific databases have always been with us . . .

Two that I am familiar with:

Daytona (AT&T): “Daytona is a data management system, not a database”. Built on top of the unix file system, this toolkit is for building application-specific and highly scalable data stores. Is used at AT&T for analysis of 100s of terabytes of call records. http://www2.research.att.com/~daytona/

DataBlitz (Bell Labs, 1995) : Main-memory database system designed for embedded systems such as telecommunication switches. Optimized for simple key-driven queries.

But these systems are **proprietary**.

Open source is a hallmark of NoSQL

What’s new? Internet scale, cluster computing, open source . . .

Something big is happening in the land of databases

The Internet
+ cluster computing
+ open source systems

→

many more points in the database design space are being explored and deployed

Broader context helps clarify the strengths and weaknesses
of the standard relational/ACID approach.

The emerging world of databases

This classification is not Complete and is a bit fuzzy-wuzzy. For example, drawing a clear distinction between Key-value stores and Document-oriented databases is not always easy. And this is Rapidly evolving with a lot of cross-fertilization.

Often overlooked in the business-oriented hoopla: This is making BigAnalytics affordable for many scientific efforts (bioinformatics, astronomy, physics, economics,…)

- **Relational**
  - Postgres
  - MySQL
- **Key-Value stores**
  - Riak
  - Redis
  - BerkeleyDB
- **Column-oriented databases**
  - BigTable,
  - Cassandra
  - Hbase (build on Hadoop)
- **Document-oriented**
  - MongoDB
  - CouchDB
- **Graph databases**
  - Neo4j
  - VertexDB

Key-Value Stores

- Mapping Key to blob-of-byte that application must “parse”
  - Example : Riak (modeled on Dynamo, eventual consistency), Cassandra
  - Typically no “query-language” for values
- Mapping Key to “semi-structured” value
  - Example: Redis

Huge advantage: can design data representation so that all
data needed for a given update is present on a single machine.
Data can easily be partitioned (say by key ranges) over
many machines. Map-reduce initiated from set of keys . . .

Disadvantage: Data retrieved by key only. And it is hard to enforce relationships
between different values. If this is important for your applications, then perhaps
Look elsewhere …

JSON

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Tables require joins

$S(\underline{A}, B, \overset{\text{FK}}{C}) \bowtie R(\underline{C}, D, \overset{\text{FK}}{\underline{E}}) \bowtie T(\underline{E}, F)$

(FK = Foreign Key)

A	B	C	D	E	F
A1	B1	C1	D1	E1	F1
A1	B1	C1	D2	E2	F2
A1	B1	C1	D3	E3	F3
A2	B2	C2	D4	E4	F4
A2	B2	C2	D5	E5	F5
. . .	. . .				
. . .					

- How could tables be partitioned over multiple servers?
- Enforcing referential integrity is VERY difficult in a distributed database

The Key-value approach

$S(\underline{A}, B, \overset{FK}{C}) \bowtie R(\underline{C}, D, \overset{FK}{\underline{E}}) \bowtie T(\underline{E}, F)$ (FK = Foreign Key)

A	B	C	D	E	F
A1	B1	C1	D1	E1	F1
A1	B1	C1	D2	E2	F2
A1	B1	C1	D3	E3	F3
A2	B2	C2	D4	E4	F4
A2	B2	C2	D5	E5	F5
. . .	. . .				
. . .					

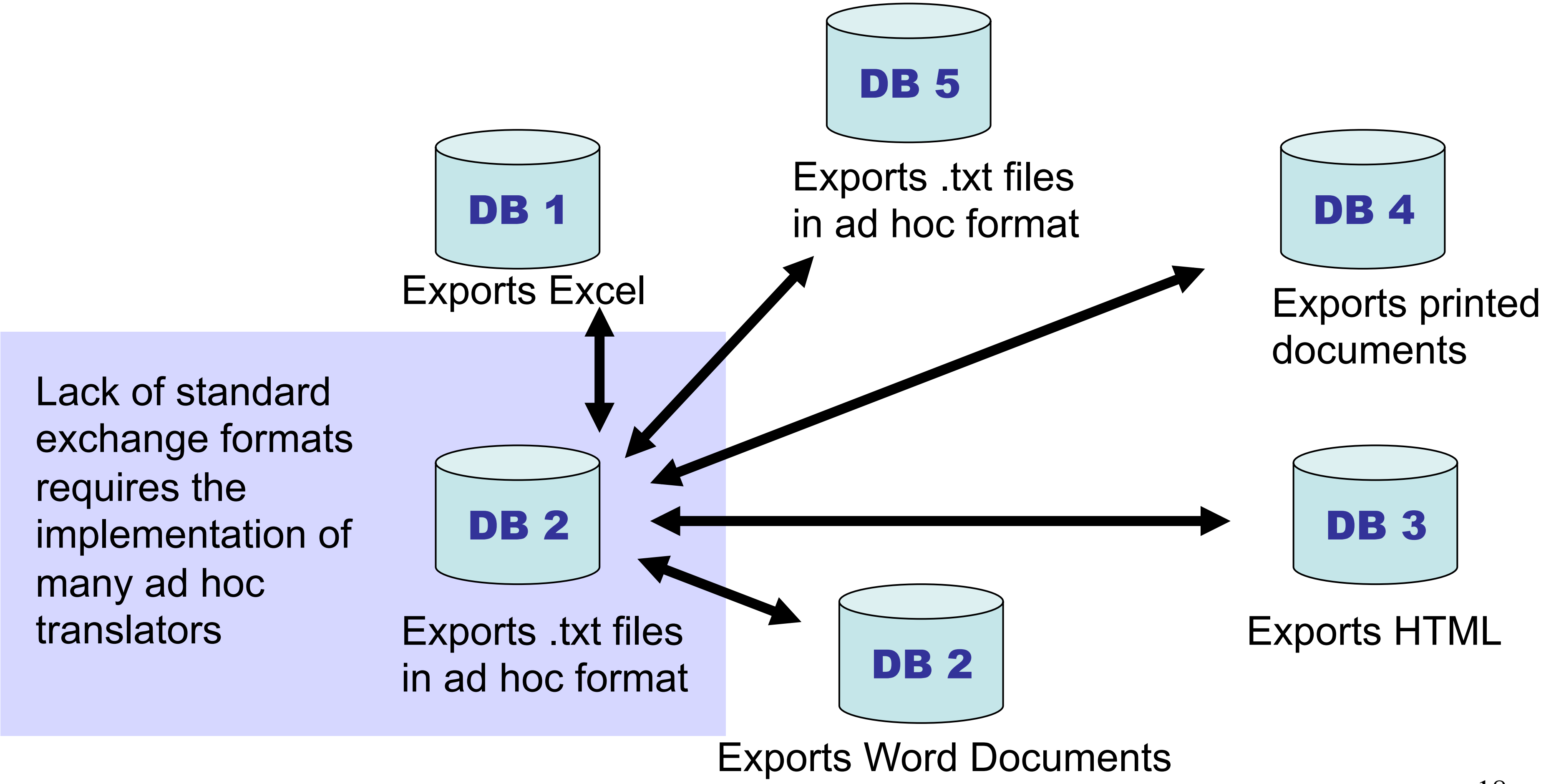
Use this instead

```
{A : A1,
 B : B1,
 stuff : [
    {D : D1, F: F1},
    {D : D2, F: F2},
    {D : D3, F: F3}
 ]
}
```

The collection of JSON objects (keyed on A) is horizontally partitioned (sharded) across many servers. When accessed, all of the application's data is in one object.

The “Data Publishing” Problem (Context: RDBMS)

Need to share data without exposing internal details of your database.



DB 5

Exports .txt files in ad hoc format

DB 1

Exports Excel

DB 4

Exports printed documents

Lack of standard exchange formats requires the implementation of many ad hoc translators

DB 2

Exports .txt files in ad hoc format

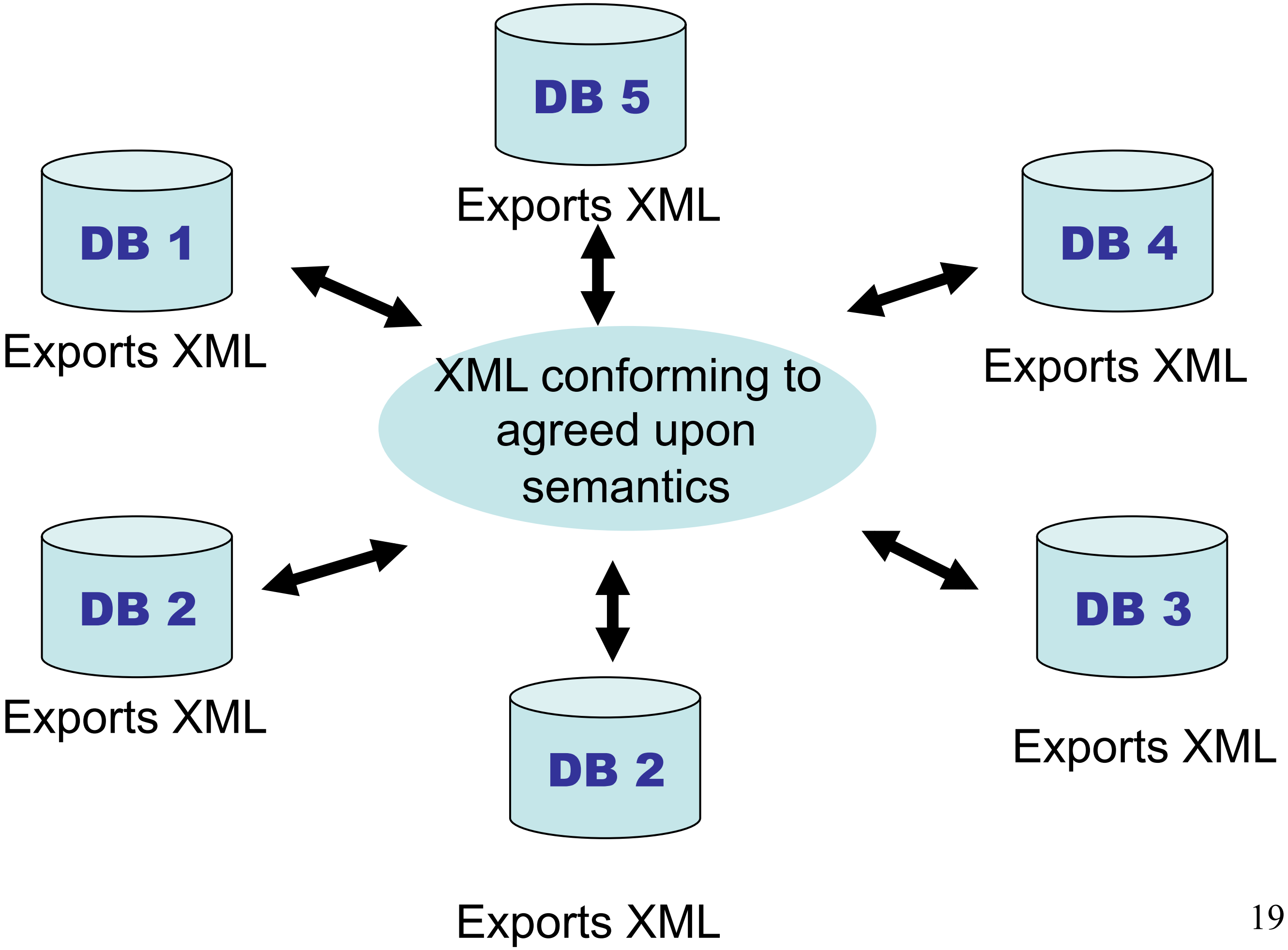
DB 3

Exports HTML

DB 2

Exports Word Documents

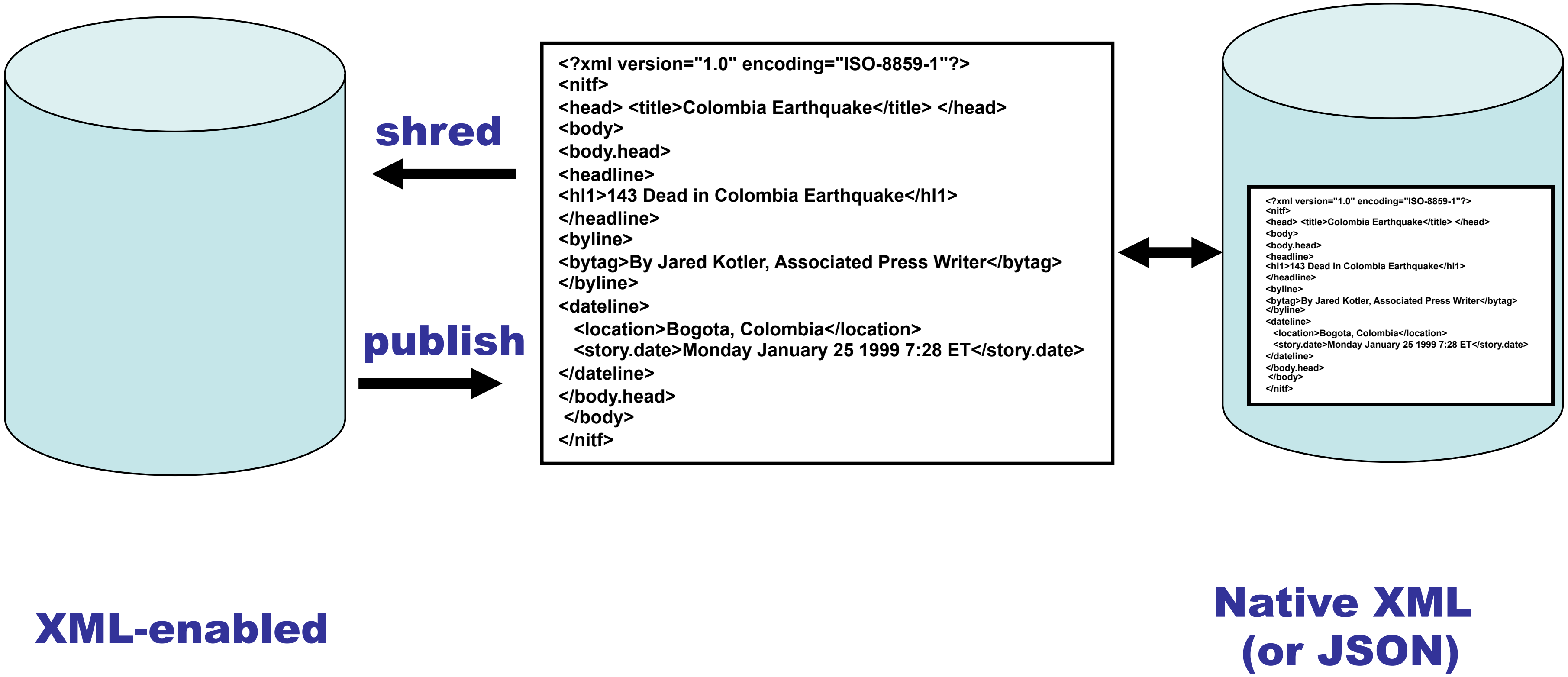
XML (or JSON) as a data exchange format



Domain specific DTDs

- There are now lots of DTDs that have been agreed by groups, including
  - WML: Wireless markup language (WAP)
  - OFX: Open financial exchange
  - CML: Chemical markup language
  - AML: Astronomical markup language
  - MathML: Mathematics markup language
  - SMIL: Synchronised Multimedia Integration Language
  - ThML: Theological markup language

Native XML (or JSON) Databases



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nitf>
<head> <title>Colombia Earthquake</title> </head>
<body>
<body.head>
<headline>
<hl1>143 Dead in Colombia Earthquake</hl1>
</headline>
<byline>
<bytag>By Jared Kotler, Associated Press Writer</bytag>
</byline>
<dateline>
  <location>Bogota, Colombia</location>
  <story.date>Monday January 25 1999 7:28 ET</story.date>
</dateline>
</body.head>
 </body>
</nitf>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<nitf>
<head> <title>Colombia Earthquake</title> </head>
<body>
<body.head>
<headline>
<hl1>143 Dead in Colombia Earthquake</hl1>
</headline>
<byline>
<bytag>By Jared Kotler, Associated Press Writer</bytag>
</byline>
<dateline>
  <location>Bogota, Colombia</location>
  <story.date>Monday January 25 1999 7:28 ET</story.date>
</dateline>
</body.head>
 </body>
</nitf>
```

Brewer’s CAP conjecture (2000)

- **C**onsistency
- **A**vailability
- **P**artition tolerance

Conjecture :
You can have at most two.

A formal proof:

Nancy Lynch and Seth Gilbert,
“Brewer's conjecture and the feasibility
of consistent, available, partition-tolerant web services”,
ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51-59.

But what do the CAP terms really mean? There seems to be no consensus . . .

Random samples of various definitions found in the literature …

- **C**onsistency
  - The system can guarantee that once you store a state in the system, it will report the same state in every subsequent operation until the state is explicitly changed by something outside the system.
  - Is equivalent to having a single up-to-date copy of the data
- **A**vailability
  - All clients can find some replica of the data, even in the presence of failures
  - A guarantee that every request receives a response about whether it was successful or failed
- **P**artition tolerance
  - The system properties hold even when the system is partitioned
  - The system continues to operate despite arbitrary message loss or failure of part of the system

Pick any two?

- **C**onsistency
- **A**vailability
- ~~**P**artition tolerance~~

Traditional relational DBMS using 2-phase commit : very good at C and struggle with A (at scale)

- **C**onsistency
- ~~**A**vailability~~
- **P**artition tolerance

Quorum/majority algorithms

- ~~**C**onsistency~~
- **A**vailability
- **P**artition tolerance

Web caching, DNS.