Scalable Data and Software Architectures – Getting Past the Hype

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

John Klein
Ian Gorton

30 Oct 2013

© 2013 Carnegie Mellon University

Copyright 2013 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM-0000711

Outline

State of the practice

Scale – Dimensions, qualities, approaches

Horizontal Scaling – Definitions, implications

NoSQL – A technology, not a solution

LEAP4BD – A system-specific risk reduction approach

Scalability – State of the practice
"The problem is not solved"

Building scalable systems is hard

- Healthcare.gov
- Netflix – Christmas Eve 2012 outage
- Amazon – 19 Aug 2013 – 45 minutes of downtime = $5M lost revenue
- Google – 16 Aug 2013 - homepage offline for 5 minutes
- NASDAQ – June 2012 – Facebook IPO

Building scalable systems is expensive

- Google, Amazon, Facebook, *et al.*
  - More than a decade of investment
  - Billions of $$$
- Many application-specific solutions that exploit problem-specific properties
  - No such thing as a general-purpose scalable system
- Cloud computing lowers cost barrier to entry – now possible to fail cheaper and faster

Scale – Dimensions

Workload

- # of concurrent sessions and operations
- Operation mix (create, read, update, delete)
- Read – query mix
- Generally, each system use case represents a distinct workload

Data Sets – Volume, Velocity, Variety

- Number of records
- Record size
- Record structure (e.g., sparse records)
- Homogeneity/heterogeneity of structure/schema

Elasticity

- Runtime peaks and valleys – how frequently, how quickly, how much

Characterizing Scalability

Who doesn’t want a scalable system? But what does that mean?

Scalability = Dependability at Scale

- Criteria is often “cost increases linearly as <X> increases”

Scalability is a composite quality that includes

- Throughput
- Latency
- Availability
- Consistency
- Security
- and more

Need to define this for each system

- The challenge is that the qualities are not independent – requirements must account for tradeoffs as system scales
- Quality Attribute Scenarios are one useful tool

“Vertical” Scaling

Scale up – “big iron”

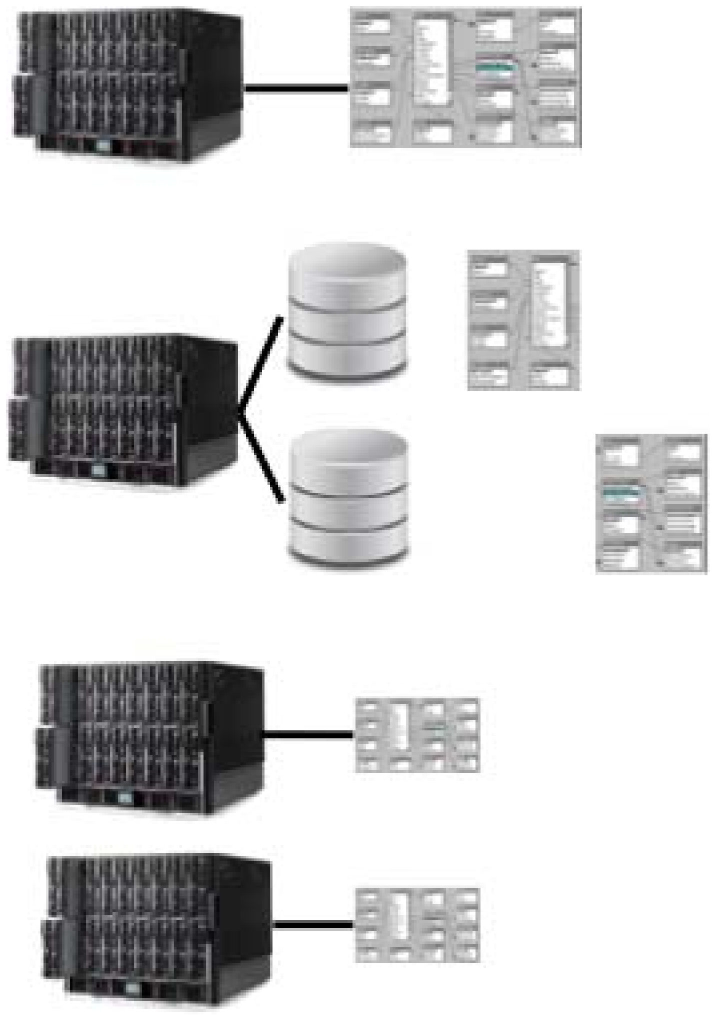
- Monolithic compute resource
- Shared disk
- More load = bigger processor and disk

Partitioned databases on disk

- Optimizes the data placement on separate files or disks
- Monolithic compute resource

Separate database instances

- Partition database across database engine instance
- Functional partitioning common (e.g., customers, orders, stock)
- More compute, more license costs



“Horizontal” Scaling

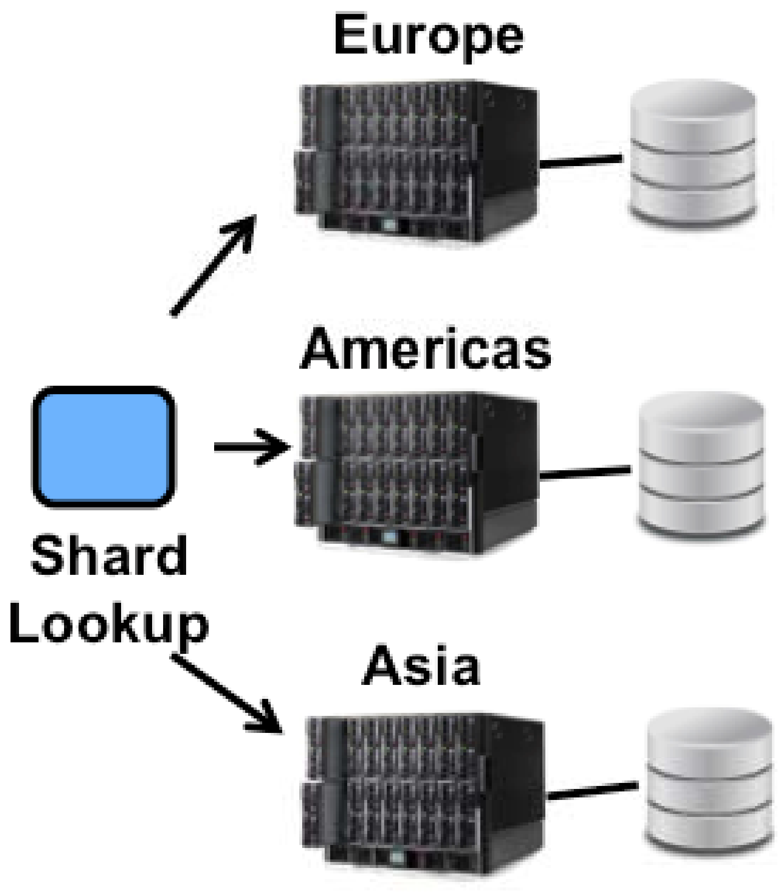
Spread data and workload across large large clusters of commodity-class hardware

For example, data sharding

- Horizontal data partitioning
- Many possible partitioning schemes, e.g. value based (region, customer ID) or a arbitrary (hashing)

Issues:

- Evenly distributing read load, write load, and data volume
- Handling shard failures



Shard - A single instance of a database, typically hosted on a commodity server, which houses a subset of the system's total dataset. Each shard within a system typically stores the same type of data, although the actual data on each shard is unique to that shard.

Horizontal Scaling Distributes Data

Distributed systems theory is hard but well-established

- Lamport’s “Time, clocks and ordering of events” (1978), “Byzantine generals” (1982), and “Part-time parliament” (1990)
- Gray’s “Notes on database operating systems” (1978)
- Lynch’s “Distributed algorithms” (1996, 906 pages)

Implementing the theory is hard, but possible

- Google’s “Paxos made live” (2007)

Introduces fundamental tradeoff among “CAP” qualities

- Consistency, Availability, Partition tolerance (see Brewer)
- “When Partition occurs, tradeoff Availability against Consistency, Else tradeoff Latency against Consistency” (PACELC, see Abadi)

CAP Tradeoffs in Modern Data-Intensive Systems

"Traditional" vertically-scaled relational database system provides

- ACID operations – atomic, consistent, isolated, durable
- "Single system image" programming model
  - application designer does not know how data is distributed
  - strong consistency model
  - operations either succeed or fail

Horizontally-scaled systems usually relax consistency to improve availability and performance

- BASE – BAsically available, Soft state, Eventually consistent
- ACID 2.0 – operations are Associative, Commutative, and Idempotent
- Application logic must understand failure modes and handle data inconsistency

© 2013 Carnegie Mellon University

Primer on data consistency models

Strong – After an update completes, all reads return the updated value

Weak – After an update completes, there is no guarantee a read will return the updated value. The period between the update and when any read will return the updated value is called the *inconsistency window*.

- Eventual – If no new updates are made, eventually every reader will see the last updated value. The DNS (Domain Name System) is eventually consistent.
  - Causal – If A communicates to B that it has updated an item, then subsequent reads by B return the updated value, and new updates by A overwrite old values.
  - Read-your-writes – If A updates an item, then reads by A return the updated value.
  - Monotonic read – If a reader has seen an update, then subsequent reads will never return older values.
  - Monotonic writes – Updates by a process are applied in the order sent.

See Vogels, "Eventually Consistent", http://queue.acm.org/detail.cfm?id=1466448.

NoSQL – Horizontally-scalable database technology

NoSQL - Originally implied “No SQL”, but stance has moderated over time

Designed to scale horizontally and provide high performance for a particular type of problem

- Most originated to solve a particular system problem/use case
- Later were generalized (somewhat) and most are available as open-source packages

NoSQL Databases – Common Characteristics

No relational schema

- De-normalized data models (application manages redundancy)
- No join queries (use an aggregation framework like MapReduce instead)
- Dynamic schema – no enforcement by database, all enforcement in application

Simple physical data models – less mismatch between programming language constructs and database constructs

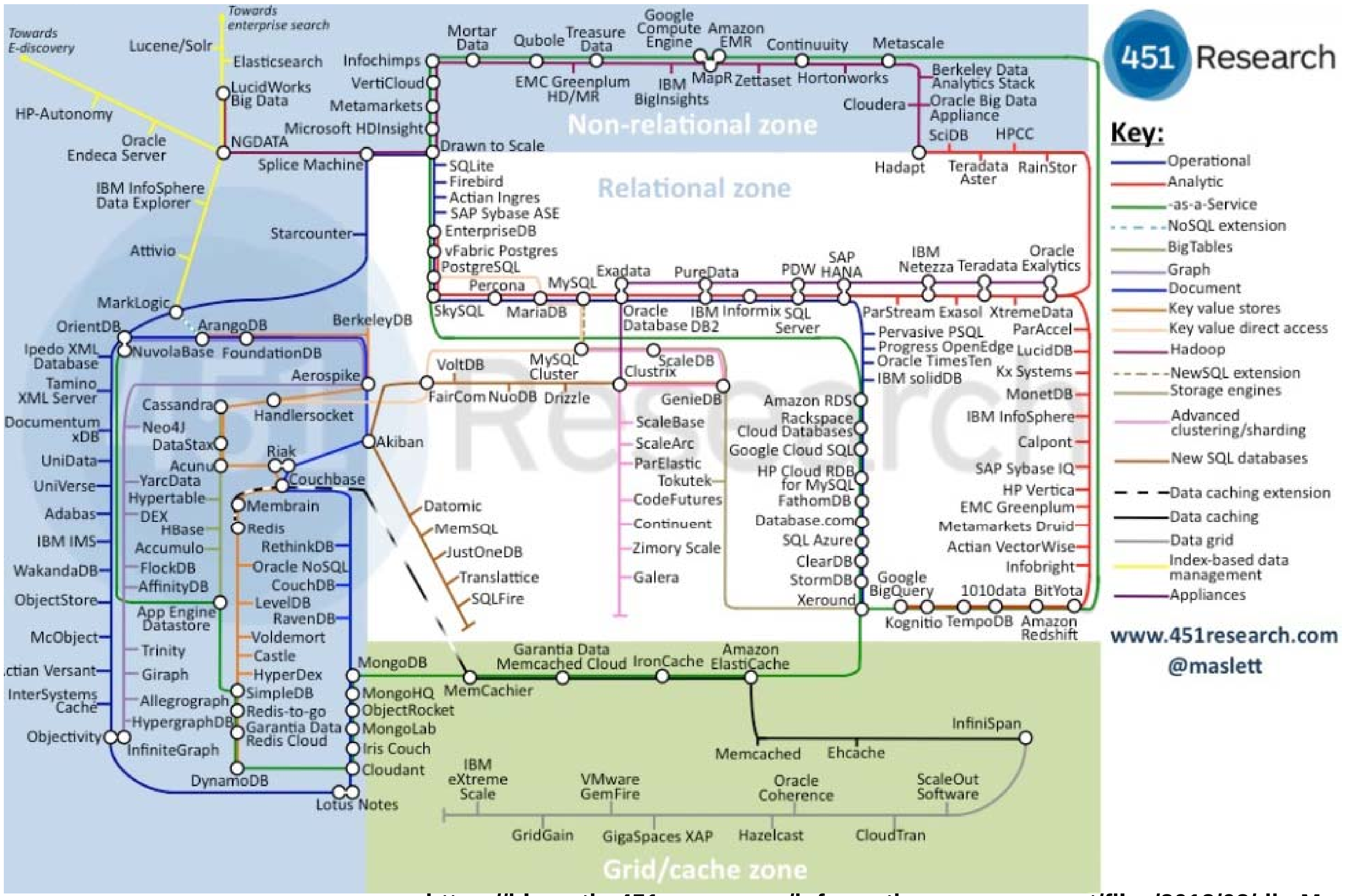
Built-in, automatic distribution

- sharding (partitioning across nodes)
- replication (copying across nodes)

Application controls durability and consistency at the operation level

- Coordinate write and read settings to achieve desired quality

NoSQL Landscape



https://blogs.the451group.com/information_management/files/2013/02/db_Map_2_13.jpg

NoSQL Landscape – Simplified to Four Types

Key-value Store

- Simple hash table – associates unique key with a piece of un-typed data
- Some implementations add metadata to facilitate queries and versioning to resolve conflicts
- Examples – Dynamo, riak, Redis, Oracle NoSQL

Column Store

- Inverted key-value store – indexed and stored by values, keys not unique
- Inbox search use case - key is message ID, value is keyword - “Find all messages that contain ‘System Engineering’”
- Examples – Cassandra, Accumulo, HBase, Amazon SimpleDB

Document Store

- Key-value store where “value” is a document (XML, JSON, BSON, …)
- Often includes full-text indexing to query within a document
- Examples – MongoDB, Couchbase

Graph Store

- Records represent nodes and edges/links – properties attached to each store the data
- Links are traversable, concepts like “next-to” or “friend-of-friend” are built-in
- Example – Neo4J

Polyglot Persistence

Evolution from Relational Databases to NoSQL Databases

Relational Paradigm – “One size fits all”

- Normalize your data model and wedge it into a relational schema
- Tune the database for your problem
- Simpler applications using single system image abstraction
- Usually good enough and easier than any alternative

NoSQL Paradigm – “Mix and match”

- Break up your persistence requirements and choose the best database for each part of your problem
- Several different databases technologies in the system – polyglot persistence
- Usually better performance
- Simpler applications for simple problems (compared to using relational store)
- More complex applications for other problems (need to re-create some of the SQL capabilities in the application)

Why does it matter?

A (sweeping) software engineering practical perspective

- Software and data architect are two different roles, each with separate concerns
- Roles separated by the single system image abstraction
  - Software architects live above
  - Data architects live below
- Driven by different, deep technology specializations

NoSQL leads us to architecture approaches that blur this separation

- Simpler data models - less specialization required
- Implemented as highly distributed systems, with a significant part of the architecture defined by the data management systems is use

NoSQL is just one component of scalable data-intensive system solution

NoSQL Technology and Architecture Selection

Systematic approach is needed to evaluate technology and architecture alternatives

Evaluation context:

- Rapidly changing technology landscape – balance speed with precision
- Large potential solution space – need to quickly narrow down
- Scale makes full-fidelity prototyping impractical
- Technology is highly configurable – need to focus on go/no-go criteria and avoid trap of optimizing every test run

SEI has developed a risk reduction approach called *Lightweight Evaluation and Architecture Prototyping for Big Data (LEAP4BD)*

- Assess the system context and landscape
- Identify the architecturally-significant requirements and develop decision criteria
- Evaluate candidate technologies against quality attribute decision criteria
- Validate architecture decisions and technology selections through focused prototyping

LEAP4BD Outcomes

Focus on the data storage and access services addresses the major risk areas for an application

- Keeps the method lean, rapidly produces design insights that become the basis for downstream decisions

Highly transparent and systematic analysis and evaluation method significantly reduces the burden of justification for the necessary investments to build, deploy, and operate the application

- Data-driven decisions

Informed adoption of modern technologies to reduce costs while ensuring that an application can satisfy its quality attribute requirements

Increased confidence in architecture design and database technology selection

- Hands-on experience working with the technology during prototype development, reduces development risks

Identifies risks that must be mitigated in design and implementation, along with detailed strategies and measures that allow for continual assessment

LEAP4BD Status

Currently piloting with a federal agency

- Project involves both scalable architecture design and focused NoSQL database technology benchmarking, as well as an assessment of features to meet the key quality attributes for scalable big data systems

We are interested in working with organizational leaders who want to ensure appropriate technology selection and software architecture design for their big data systems

See http://blog.sei.cmu.edu/post.cfm/challenges-big-data-294 for more details

Wrap-up

Building scalable data-intensive systems is NOT a solved problem.

Avoid the buzzword “scalable” – be specific about

- Dimension – workload, data volume/velocity/variety, and elasticity
- Qualities and tradeoffs – throughput, latency, availability, consistency, security, etc.

Architectures that use horizontal scaling are the current best solution for many scalable systems

- Partition computing and storage across large numbers of commodity-class hardware nodes
- These are distributed systems, with all of the accompanying benefits and challenges

NoSQL databases are designed for horizontal scaling

- Simpler database (compared to relational) with built-in data distribution

LEAP4BD is a systematic approach to evaluate technology and architecture approaches

Contact Information

John Klein

Senior Member of the Technical Staff

Software Systems Division

Telephone: +1 412-268-4553

Email: jklein@sei.cmu.edu

U.S. mail:

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

World Wide Web:

www.sei.cmu.edu

www.sei.cmu.edu/contact.html

Customer Relations

Email: customer-relations@sei.cmu.edu

Telephone:	+1 412-268-5800
SEI Phone:	+1 412-268-5800
SEI Fax:	+1 412-268-6257