Data Centric Computing for Internet Scale Enterprises

Yuqing Gao

Seetharami Seelam, Xavier Guerin, Wei Tan, Yanbin Liu,
Liana Fong, Paolo Dettori

IBM T. J. Watson Research Center

Big Data is Real

"Every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone."

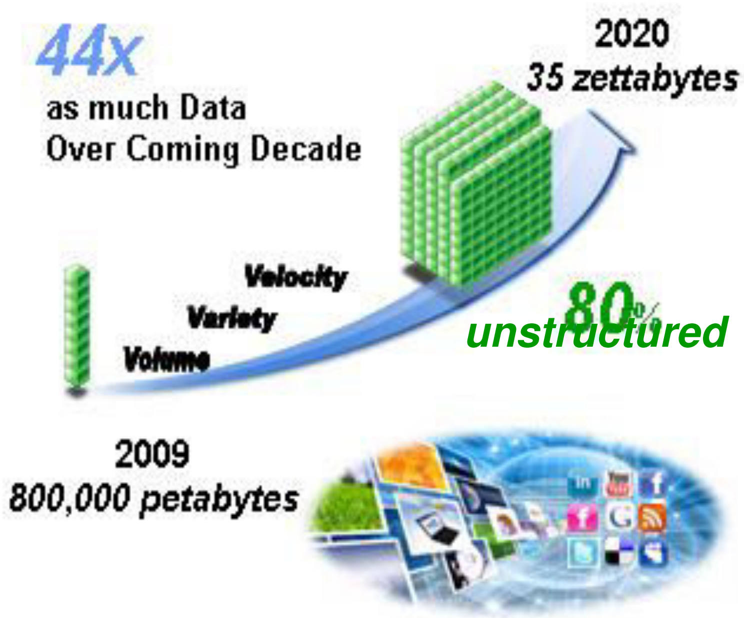
Source: IBM Website

Source: McKinsey Global Institute analysis

Where are the data from?

- Mobile devices and sensors
  - 5.6 billion mobile phones in use in 2011
- Social networks
  - Facebook, Twitter, Linkedin
- Online service
  - Google, Yahoo!, Amazon
- New sciences
  - Life sciences

"Big Data"? - Extracting insight from an immense volume, variety and velocity of data, beyond what was previously possible



Variety – Manage the complexity of multiple relational and non-relational data types and schemas

Velocity – Streaming data and large volume data movement

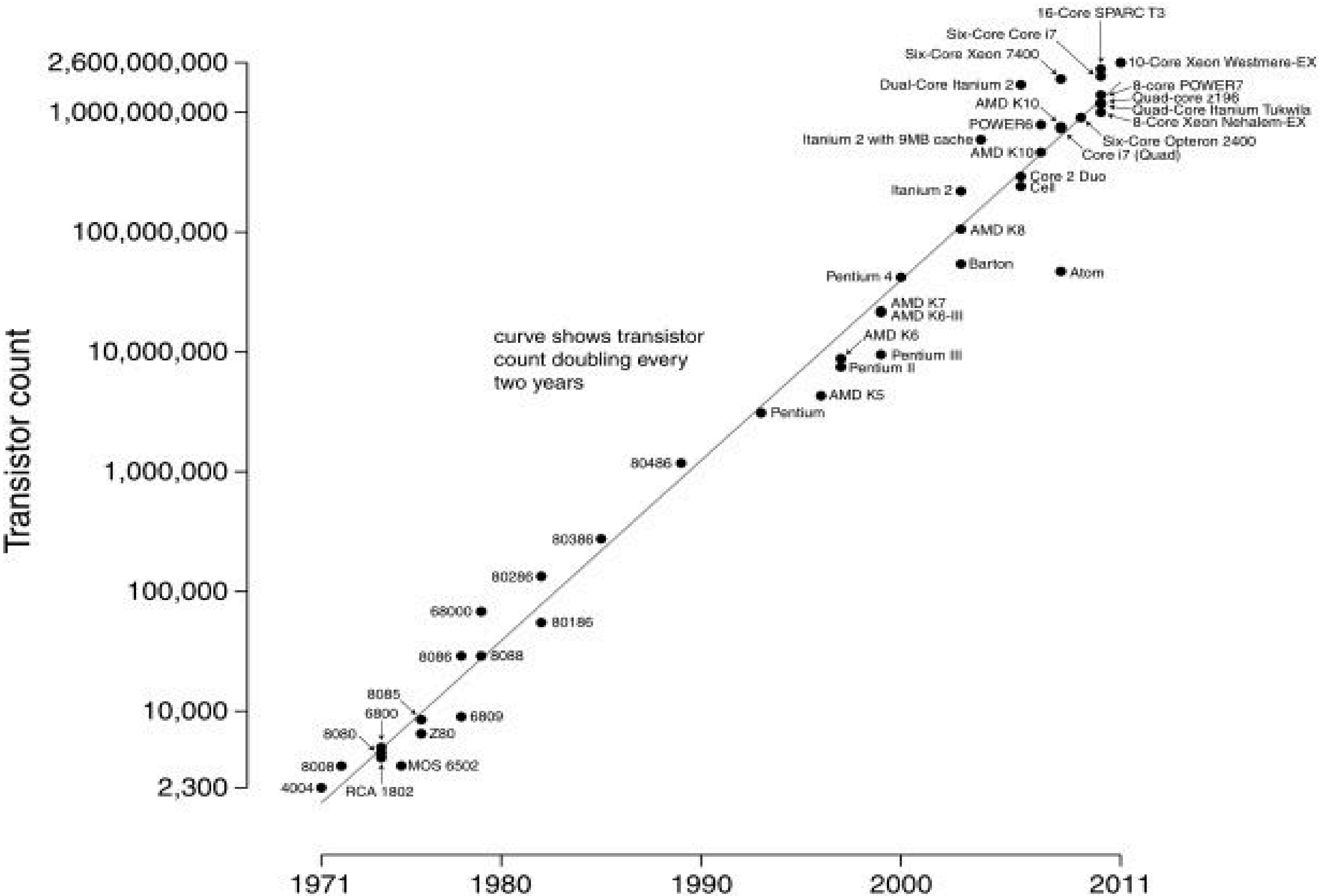
Volume – Scale from terabytes to zettabytes

Information Overload

- Organizations recognize they need powerful alternatives **beyond traditional SQL database** technology to manage, process, and leverage Big Data for Business advantage
  - *Traditional SQL databases store data in form of schemas which presents a challenge when managing, processing, and analyzing unstructured data*
  - *Big Data is about processing and storing as fast and as efficiently as possible*

Sources: IDC Digital Universe,

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Polyglot: the buzz

Java has ruled the enterprise application space for 15+ years but its dominance is cracking giving birth to …

Polyglot: No One Language Will Rule the Cloud

Modern Web apps simply refuse to be homogenized.

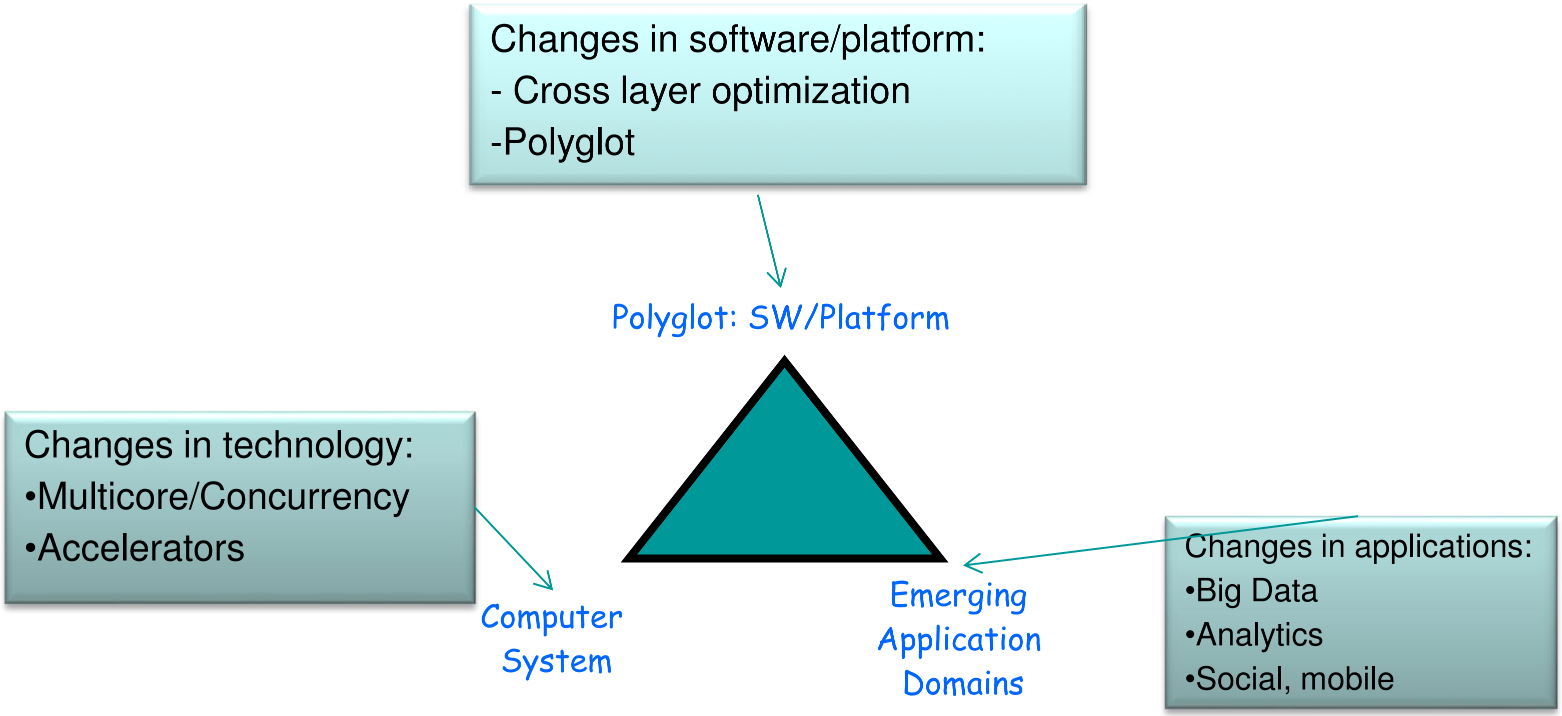
the reason you choose to go polyglot… is because you' re [always] choosing the best tool for the job

Businesses that already have the computing power are starting to consider scrapping their current application server architectures altogether, and to host their own Heroku-like platforms internally.

emergence of versatile new "polyglot programmers"

Polyglot: future programmers, future platforms

The Perfect Storm?



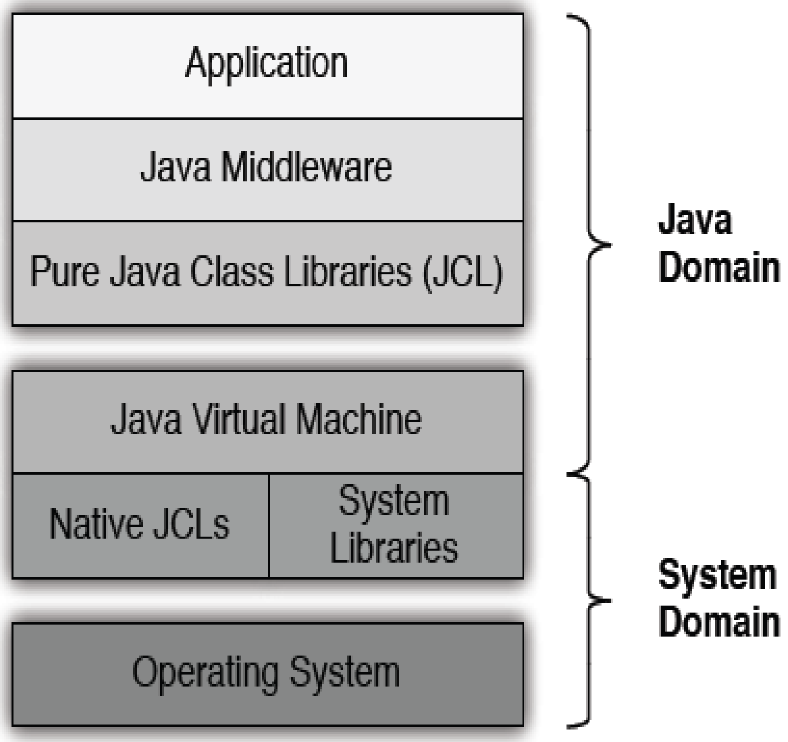
Outline

- Parallelization issue of data intensive enterprise java workloads on multicore systems
- Design requirement of data centric computing systems
- Global secondary index for HBase (Hadoop Database)
- RDMA Enabled Ultra Low Latency Distributed Cache
- Polyglot runtime systems & challenges for benchmarking community

Evaluation of Multi-Core Scalability Bottlenecks in Enterprise Java Workloads *(MASCOT'2012, X. Guerin, W. Tan, Yanbin Liu, S. Seelam and P. Dube)*

Motivations

- **Multi-core designs are replacing high-frequency operating architectures**
- **Enterprise Java applications not able to fully exploit multi-core parallelism**
- **Evaluate scalability and parallelization bottlenecks in each layer of Java application's stack, provide solutions and identify commonalities**
- **Simply run known benchmarks are not enough**



Methodology

- **Evaluate applications on a representative multi-core machine:**
  - 16 core IBM Power7 system
- **Conduct analysis using a top-down methodology based on good-faith**
  - *Each layer of rank n (topmost) of the software stack was profiled with the hypothesis that each other layer n−1 . . . 1 is scalable and free of lock contention until the last layer has been reached.*
- **Execute the chosen applications on the maximum available number of cores (16) and increase the number of application threads from 1 to 16**
  - Each application thread run on its own processor core.
  - Compare with *perfect scalability*
    - Perfect scalability entails linear throughput increase and constant latency up to hardware limits
  - Discover and analyze bottlenecks using tools including
    - Light-Weight Java Trace tool
    - WAIT and JProfiler,

Evaluation Environment

- **Hardware Environment: IBM POWER7-based blade**
  - 16 Power7 cores on two POWER7 sockets
  - 256GB system memory
- **Software Environment**
  - SUSE Linux Enterprise Server 11
  - IBM's J9 Java Virtual Machine
    - Allocate sufficient amounts of memory heap to avoid garbage collection activity (GenCon garbage collection policy)

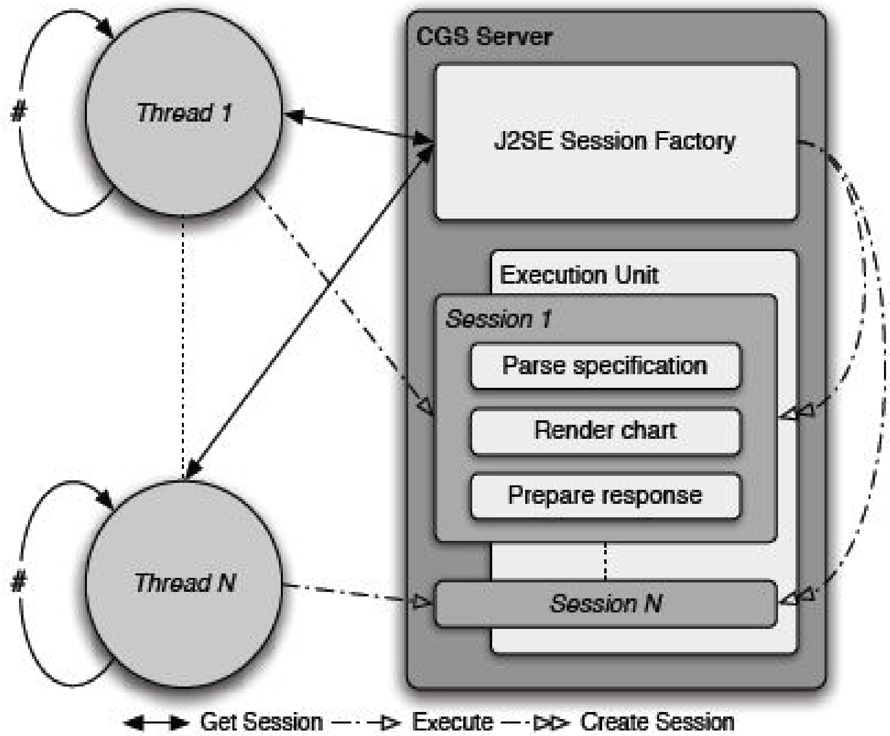
Enterprise workload evaluated

- **ILOG Business Rules Management System**
- **DayTrader PDF Document Generation**
- **IBM Cognos Chart Generation Service**
- **Many customers' application programs**
- **…**

IBM Cognos Chart Generation Service

- **IBM Cognos Business Intelligence (BI) is a software suite for enterprise-scale reporting, analysis, scorecarding and monitoring.**
- **Chart Generation Service (CGS) is a Cognos BI component that produces charts and figures to be used in various reports including PDF, Microsoft Excel and HTML**
- **Benchmark: Generate a report that aggregates the gross margin of a fictional company, categorized by product line and geographic region, and displays the result in two pie charts respectively.**
  - Query the Cognos database before hand to avoid database access during the running of the benchmark

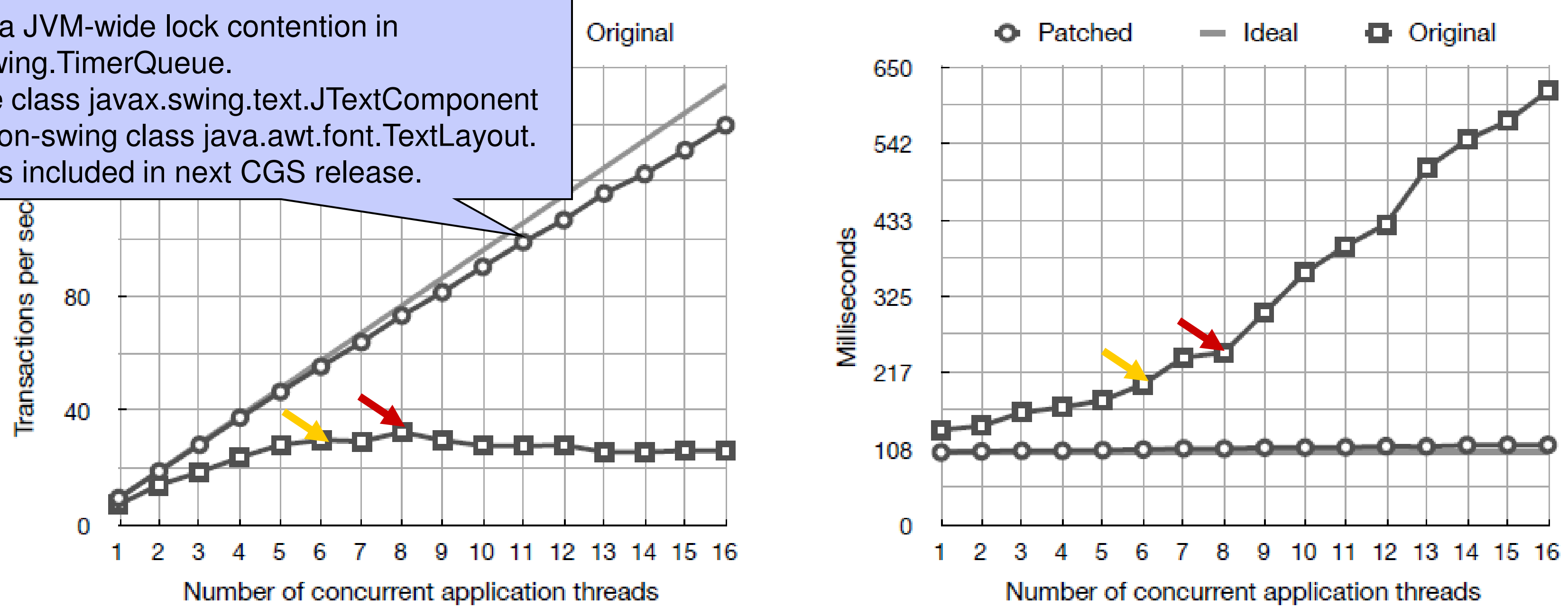
CGS Benchmark Execution



- **Arguments: 1) the number of execution threads to run in parallel; 2) the number of execution iterations for each thread**
  - A benchmark loop with the given number of iterations is instantiated for each thread.
  - The loop creates a connection with CGS, and sends an execution request with the chart specification
  - Finally, the execution threads gather the number of Transactions Per Second (TPS) as well as the average rule execution latency for its run.

CGS throughput and response time results

Identify a JVM-wide lock contention in javax.swing.TimerQueue.
Replace class javax.swing.text.JTextComponent with a non-swing class java.awt.font.TextLayout.
The fix is included in next CGS release.

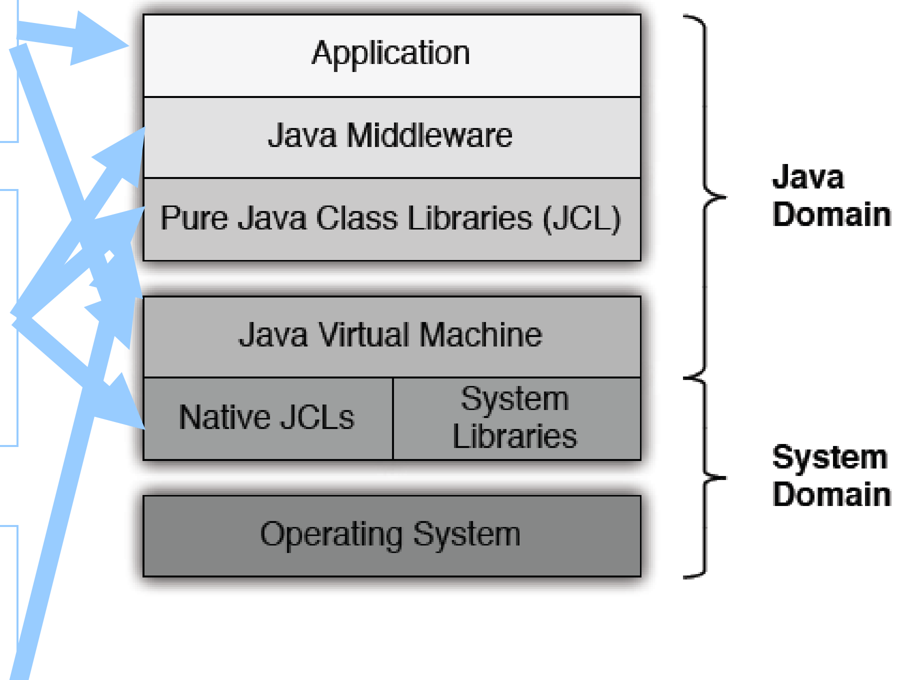


(a) Throughput

(b) Latency

Evaluation of Enterprise Java workloads

- **ILOG: Discover contention at Application and JVM layers.**
- **PDF: Discover contention at Java Middleware, JCL and Native JCLs layers**
- **Cognos CGS: Discover contention at JCL layer**

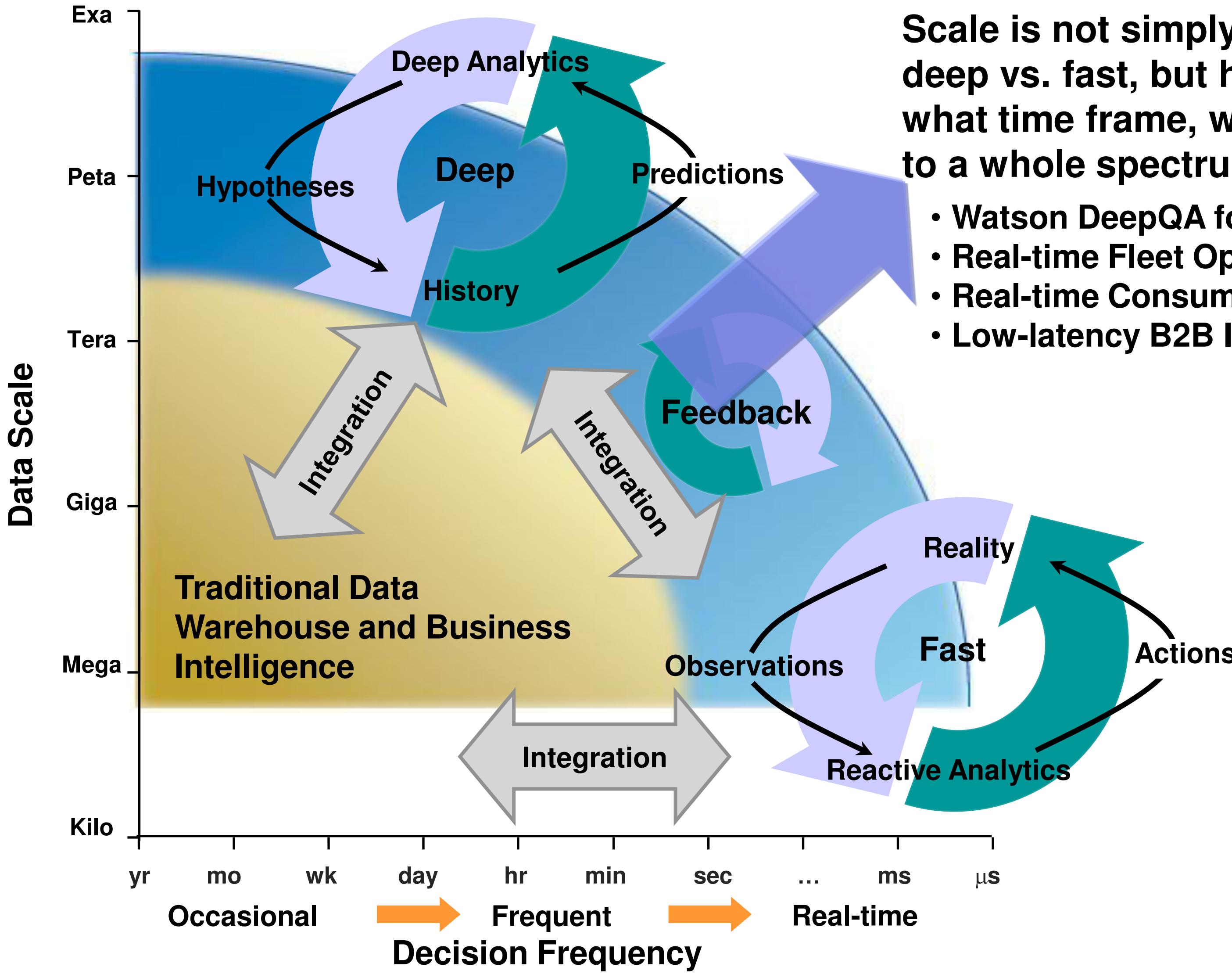


Layer Interplay:
- Application & JVM Layers
- JVM & OS & Application Layers

Outline

- Parallelization issue of data intensive enterprise java workloads on commercial multicore systems
- Design requirement of data centric computing systems
- Global secondary index for HBase (Hadoop Database)
- RDMA Enabled Cache
- Polyglot runtime systems & challenges for benchmarking community

Big Data is about analyzing data at scale along the dimensions of volume, velocity, variety, and veracity



Scale is not simply a matter of deep vs. fast, but how deeply in what time frame, which gives rise to a whole spectrum of analytics

- **Watson DeepQA for Jeopardy!**
- **Real-time Fleet Optimization System**
- **Real-time Consumer Engagements**
- **Low-latency B2B Interaction**

Key trends for big data and analytics across industries and segments (Financial, retail, government, fraud detection, healthcare, energy, etc.)

- **Internet and social media scale data**
  – Volume, velocity, variety, and veracity
  – Variety: Unstructured and data from smarter devices play a role
  – High throughput: large number of concurrent users/devices
- **Deep analytics on data at rest**
  – Finding of non-trivial relations
  – Competitive advantage
- **Low-latency analytics on massive and rapidly generating data, i.e., data in motion**
  – Timeliness in decision making
  – Interactive: client facing
- **Use of operational and transactional data for analytics:**
  – Concurrency of high velocity data acquisition and analytics on same data source
    - Need for low-latency analytics using transactional data, historical data, and internet and social media data
  – Timeliness of analytics to generate appropriate actions (e.g. promotion, fraud detection, intelligence, etc.)

Requirements

- Intelligent decision-making using: "Mobile+Cloud+Analytics+Big Data" in context of transactions
  - Analytics become embedded and pervasive

- Desire to have a large, low latency, "In Memory" model
- Focus on data parallelism with a synchronous view of data across cluster
- Large byte addressable shared in-memory pool

- Design Principles
  - Moving the compute engines into the Data
  - Making sure the DRAMs have the right Data
    - Effectively a super large cache of the applications usage of Big Data
  - More Direct Addressing of Data
    - Minimal indirect addressing of Data
  - Fewer Copying/Buffering and Replication of Data
  - Making the large amount of Big Data, Fast, Easy to Access and Easy to Manage
  - Focus on SW, use off-the-shelf HWs (before new HW is built)

Outline

- Parallelization issue of data intensive enterprise java workloads on commercial multicore systems
- Design requirement of data centric computing systems
- Global secondary index for HBase (Hadoop Database)
- RDMA Enabled Cache
- Polyglot runtime systems & challenges for benchmarking community

Categories of NoSQL Use-case Patterns

- Rapid development of web-scale solutions
  - Chosen for flexible schema
  - Web-scale apps, high-performance, read-only, not complex
  - Short life or plan to replace frequently,
  - New applications demand rapid iteration
- Scalability for web-apps
  - High ingest rates
  - Ratio of value to number of records is low: No cleansing, no ETL, no Load on ingest
  - Analyze the data where it lands
  - Semi-structured data that can be grouped on ingest
- Scalable Analytics
  - Scalable fault tolerant framework for storing and processing MASSIVE data sets (Hadoop)
  - Lower cost
  - Online update capability
  - Gives you point access to data in MR, not just sequential access
  - Records stored in distributed file system
  - Ratio of value to records is low
- Scalability for a class of current RDBMS apps: (TaoBao

Think NOSQL Agility

Think NOSQL scalability

Think NOSQL data warehousing

Global secondary index for HBase (to appear in IBM J of R&D, by L Fong, W Tan, et al)

Motivation: NoSQL stores and HBase (aka., Hadoop database)

- NoSQL is emerging -- “to be used widely during the next 5 years” [Gartner]
  - Pros:
    - Flexible schema: table, graph, object, K/V, document. On size on longer fits all.
    - Configurable consistency to deal with Internet workload
    - Scale-out horizontally on commodity HW; or hosted on cloud for easy use.
  - Cons: limited API, less mature: not “enterprise-ready” (from our SWG partner)
- Research challenges
  - Scalability, consistency, index, ACID, …
- Categories

Type	Feature	Example
Key/value	key-object mapping	Dynamo (Amazon), IBM WXS
Document	XML, JSON, BSON docs	MongoDB
Graph	social relations, road maps	neo4j
Tabular (column)	Table-like, extensible schema, convergence of operation and analytics	BigTable family (HBase, Cassandra)

Challenge: HBase has no secondary index

- Index: data structure for queries on non-primary attributes; well studied in RDBMS
  – Example (Yelp.com): *reviews* by *users* to *business*, with a *star* ratings
  – Queries need index: list all reviews of a business, user, or star

UserID	ReviewID
U01	…
…	…

Stars	ReviewID
5	…
…	…

BusinessID	ReviewID
B001	…
…	…

Need index on *User*, *Star* and *Business*

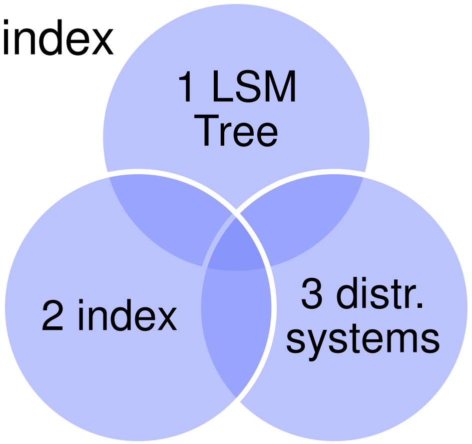
Millions of rows ←

ReviewID	Text	Stars	UserID	BusinessID
R00001	…	5	U01	B001
…	…	…	…	…

- Gap: HBase has no secondary index; query w/ table scan via MapReduce
  – Not acceptable for *ad hoc* queries

Index maintenance

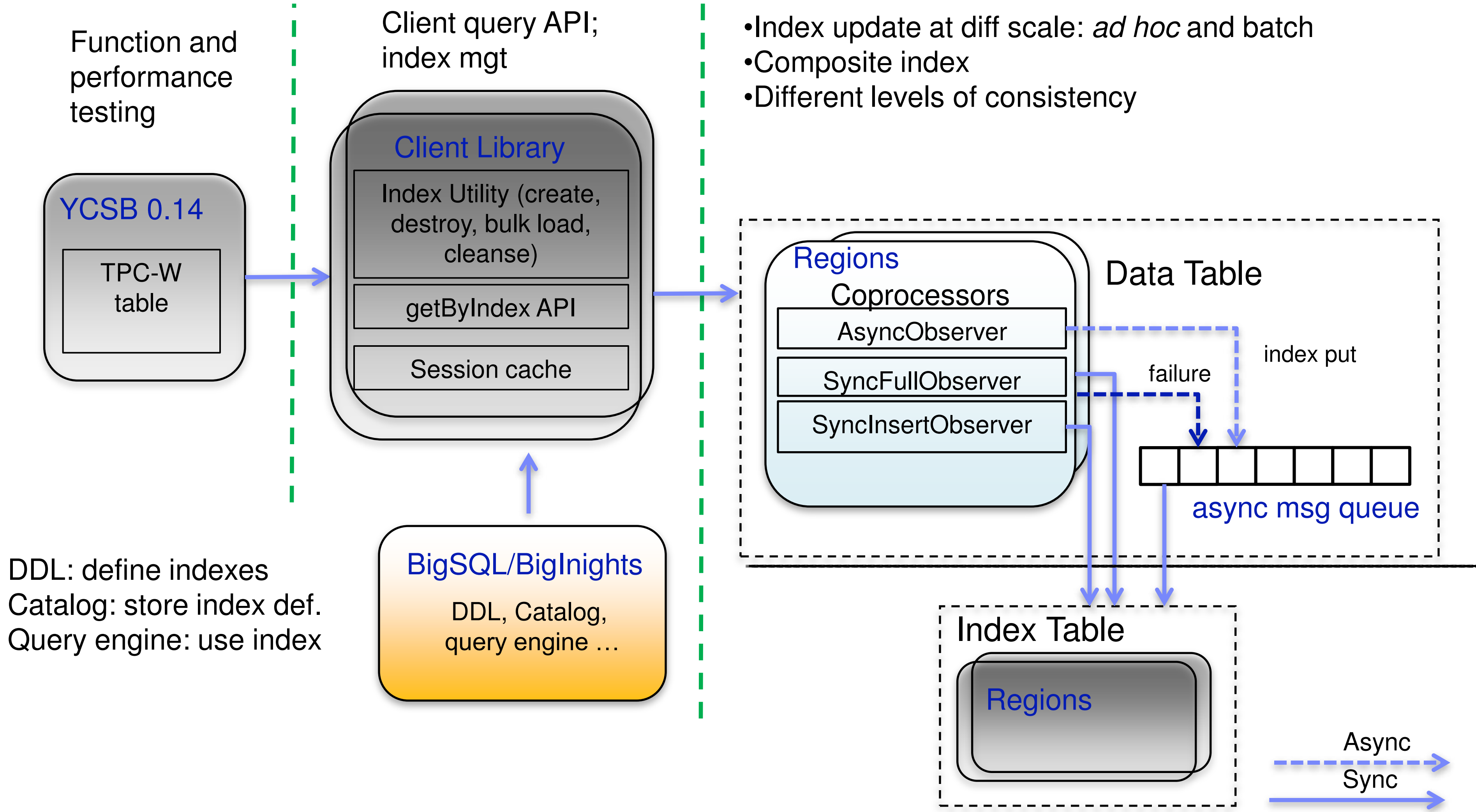
1. Log Structured Merge tree: a reviving interest in it
   a) Write workload increasing; 10~20% → > 50% (by Yahoo!)
   b) With high insertion rate: click streams, sensors, mobile…
   c) With non in-place update and slow read, index update can be slow
2. Index with high insertion rate
   a) Solutions for B+ trees and used in RDBMS, e.g., deferred index
   b) No approach systematically tackle this issue in LSM tree
3. Distributed systems
   a) Distributed index maintenance needs coordination
   b) Examine performance/consistency tradeoff



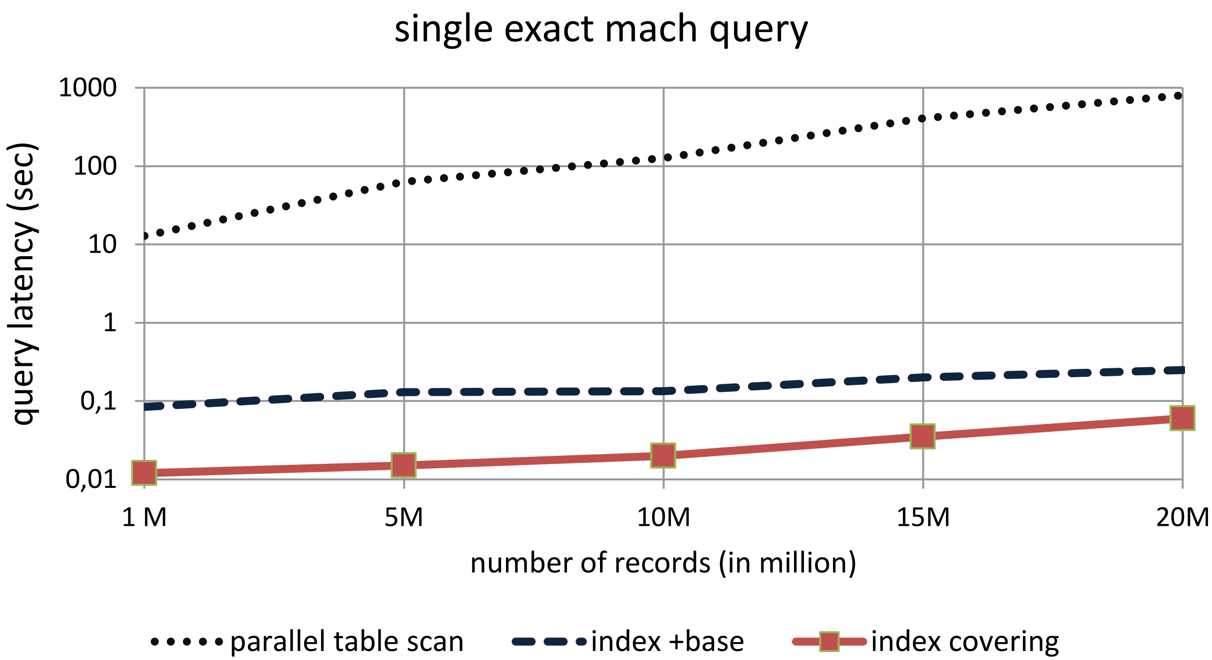
Solution:
differentiated secondary Index (Diff-Index) for HBase, a global index scheme on LSM-Tree with balanced performance

Diff-Index system: global, server-managed index with configurable schemes

- Index update at diff scale: *ad hoc* and batch
- Composite index
- Different levels of consistency



Effect of adding indexes in HBase

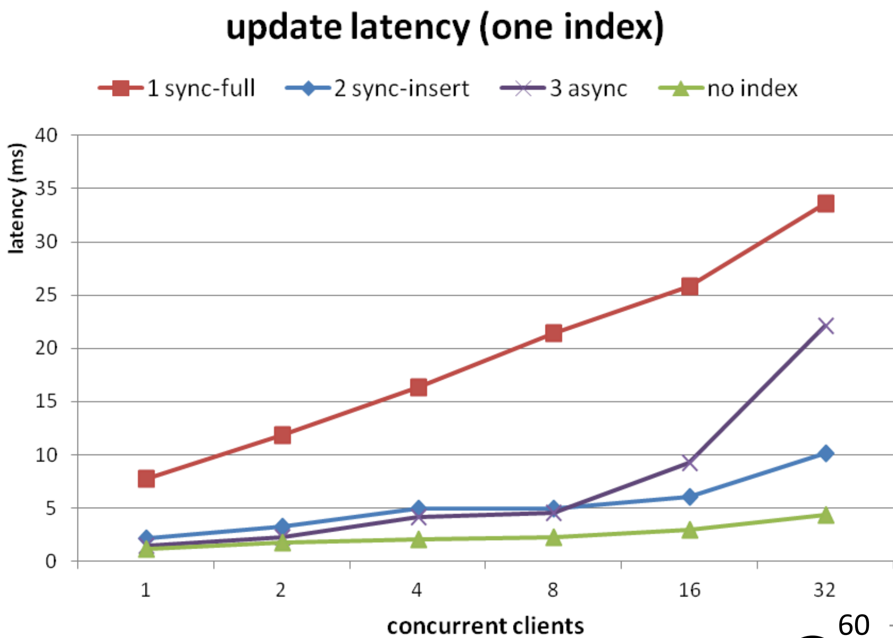


- •Parallel table scan: scan the regions in parallel
- •Index + base: combine base and index for a query
- •Index covering: index itself can serve the query

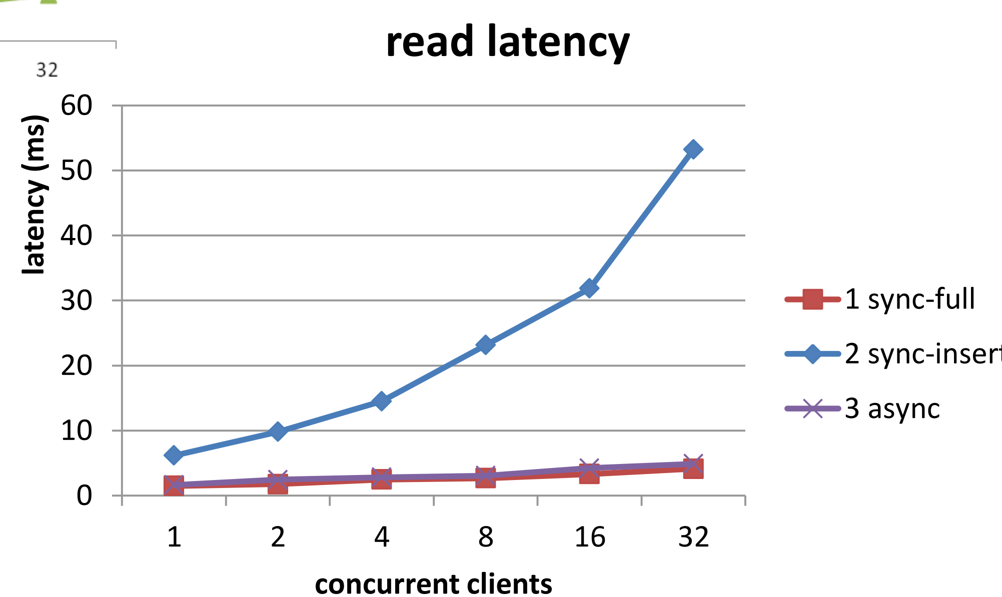
Query by index is much faster (100-1000x) and grows modestly with data size

 IBM Confidential

Performance of index update and read



You can trade read for update, or vice versa



1 sync-full — **Update slow, Read fast**

2 sync-insert — **Update fast, Read slow**

3 async — **U/R fast, inconsistent**

Outline

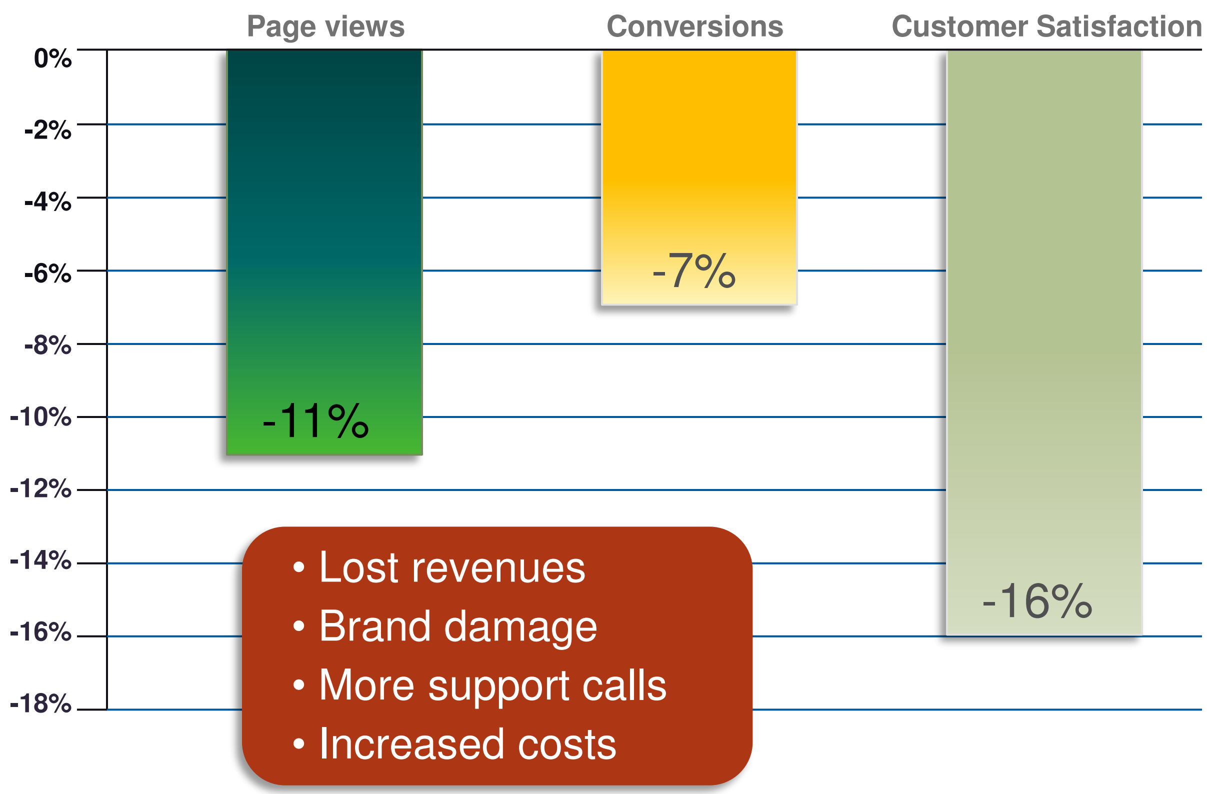
- Parallelization issue of data intensive enterprise java workloads on commercial multicore systems
- Design requirement of data centric computing systems
- Global secondary index for HBase (Hadoop Database)
- RDMA Enabled Cache
- Polyglot runtime systems & challenges for benchmarking community

WXS RDMA-Feature for Internet Scale and High Performance Enterprise Computing

- (IMPACT'2013, Yuqing Gao, Xavier Guerin, Tiia Salo)

Need for reliable speed

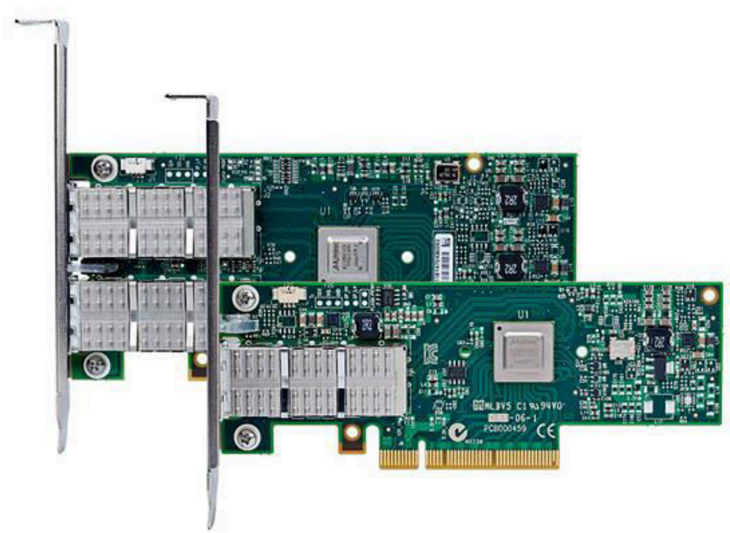
Internet response time challenges impact the revenue and customer satisfaction negatively



1. "The Performance of Web Applications: Customers Are Won or Lost in One Second," Bojan Simic, Aberdeen Group, November 2008
2. Source: Internet World Stats, Usage and Population Statistics, www.internetworldstats.com/stats.htm, December 22, 2010

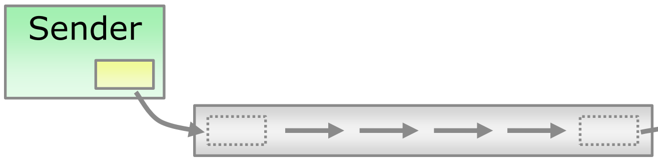
What is RDMA? Which network fabrics support RDMA?

- **Remote Direct Memory Access** (RDMA)
  - Direct access from the memory of one computer into that of another without involving either one's operating system
- **InfiniBand**
  - The original lossless low-latency RDMA fabric
  - 10/40/56Gb/s (e.g. Mellanox® ConnectX®, Connect-IB™)
- **RDMA over Converged Ethernet (RoCE)**
  - InfiniBand’s RDMA layer ported to Ethernet
  - 10/40Gb/s (e.g. Mellanox ConnectX)
- Fabric latencies in <1µs ballpark
- Up to 100km distance (e.g. Mellanox MetroX™)
  - Speed of light may become a significant factor after a few miles

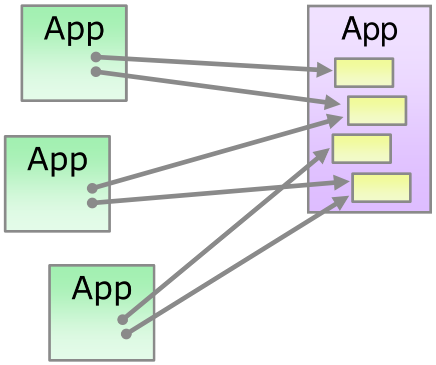


What are the common RDMA usage patterns?

- **"Faster pipe"** - the most common approach today
  - Send-receive semantics
    1. Sender copies data into a send-buffer
    2. RDMA transfer from the send-buffer to the receive-buffer
    3. Receiver copies data from the receive-buffer
  - Pros: Easy - a low hanging fruit that often can be fitted into existing apps without major rework
  - Cons: Involves CPU and copying - may not realize RDMA's full potential



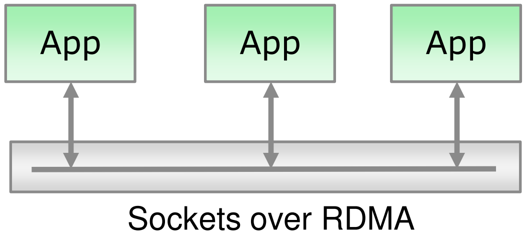
- **"Shared memory"** - mostly used in HPC
  - Pointer semantics
    1. A application hands out a set of remote pointers to it's data
    2. Peers directly read and write the data at the end of the pointers using RDMA
  - Pros: Extreme, near wire-speed performance
  - Cons: Difficult - usually requires writing the app specifically for RDMA



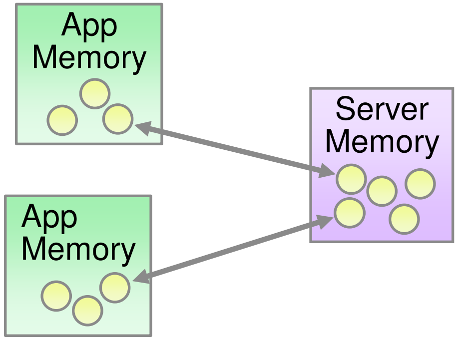
Which applications can leverage RDMA?

Three levels of RDMA exploitation

- **RDMA-optimized OS level interfaces**
  - Enable the bulk of the applications as-is with
  - A low hanging fruit with moderate overall performance improvement
  - e.g. Sockets over RDMA (JSoR)



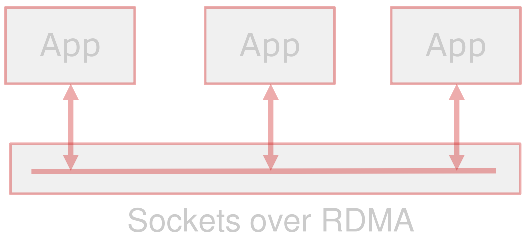
- **RDMA-optimized applications**
  - Applications that are designed for RDMA from ground up
  - An expensive approach with extreme performance
  - e.g. Trading applications



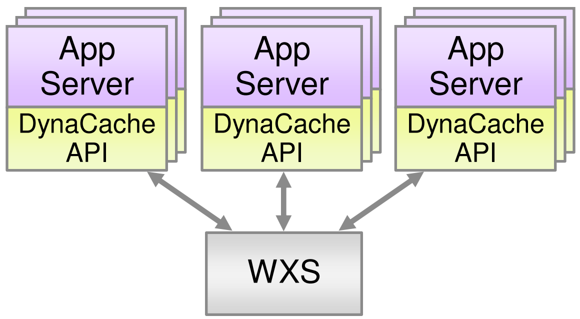
Which applications can leverage RDMA?

Three levels of RDMA exploitation

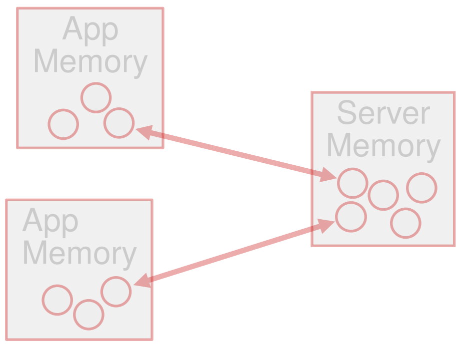
- **RDMA-optimized OS level interfaces**
  - Enable the bulk of the applications as-is with
  - A low hanging fruit with moderate overall performance improvement
  - e.g. Sockets over RDMA



- **RDMA-optimized application level interfaces**
  - Enable critical applications for scale-out
  - Substantial improvement without application code modification
  - e.g. WXS DynaCache API for caching a wide range of web application objects

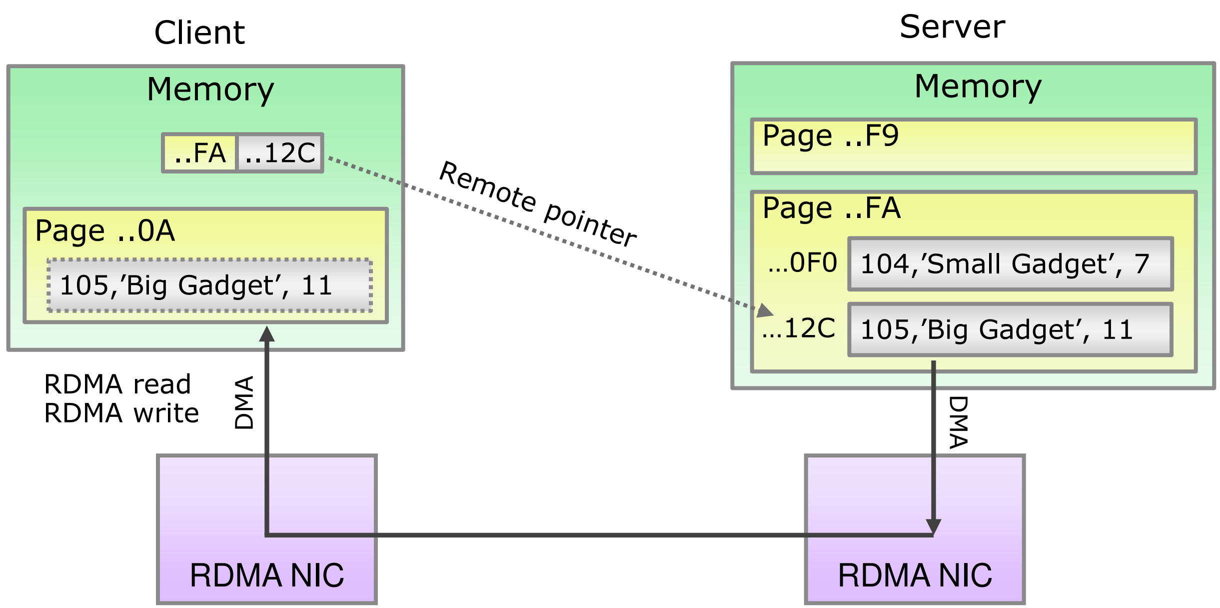


- **RDMA-optimized applications**
  - Applications that are designed for RDMA from ground up
  - An expensive approach with extreme performance
  - e.g. Trading applications



Remote pointers & one-sided RDMA

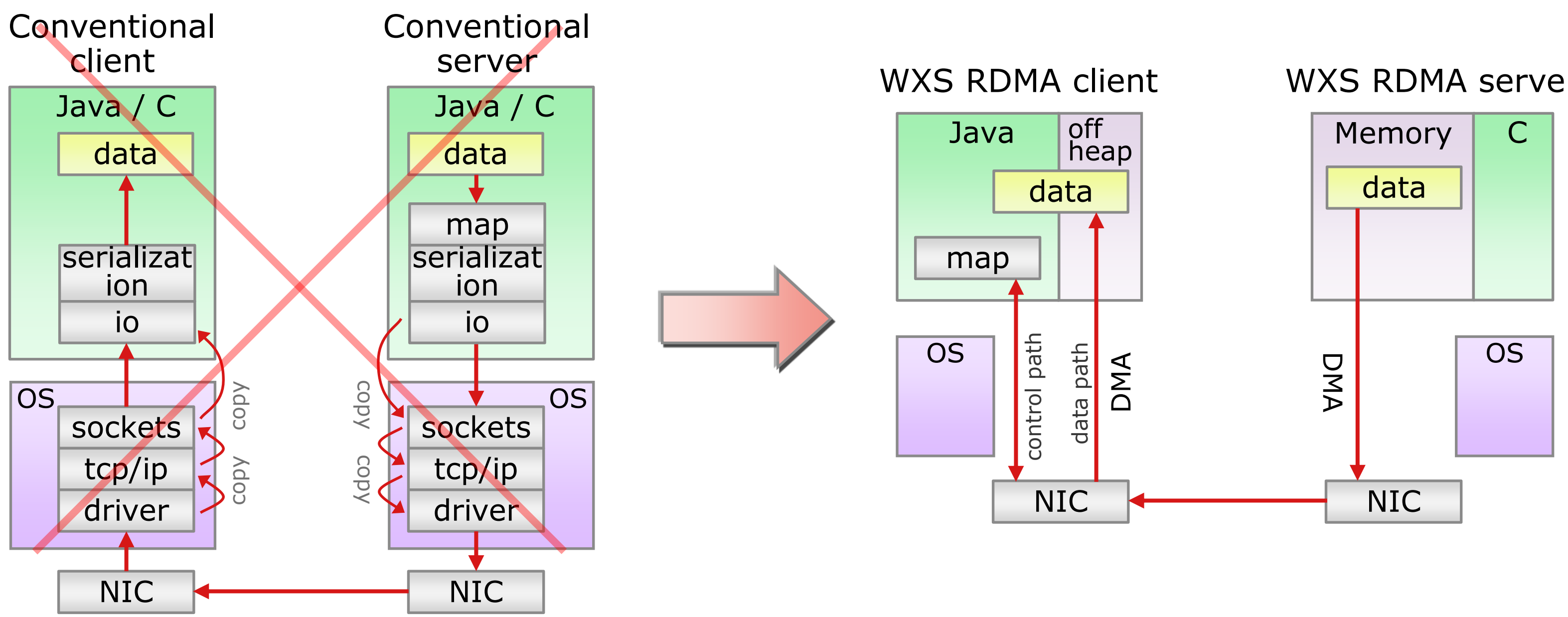
- A server can export a ***remote pointer*** that refers to a data record in a pinned & registered server memory page
- ***One-sided RDMA operations*** allow the client to directly access the remote record referenced by the pointer
  - “Remote control” the server RNIC to perform DMA to/from a memory location specified by the pointer
  - No server-side code or CPU involvement - zero server CPU utilization
- Very fast, **near wire-speed remote access**
  - Read / overwrite the remote record in single-digit microseconds



WXS and RDMA contd.

WXS and RDMA preserve the near-local access speed when scaling out

- **Ultra-low latency**
  - Substantially shorter path to shared data
  - Completely bypass the OS and network stacks



Example: WXS RDMA vs. Redis & Memcached

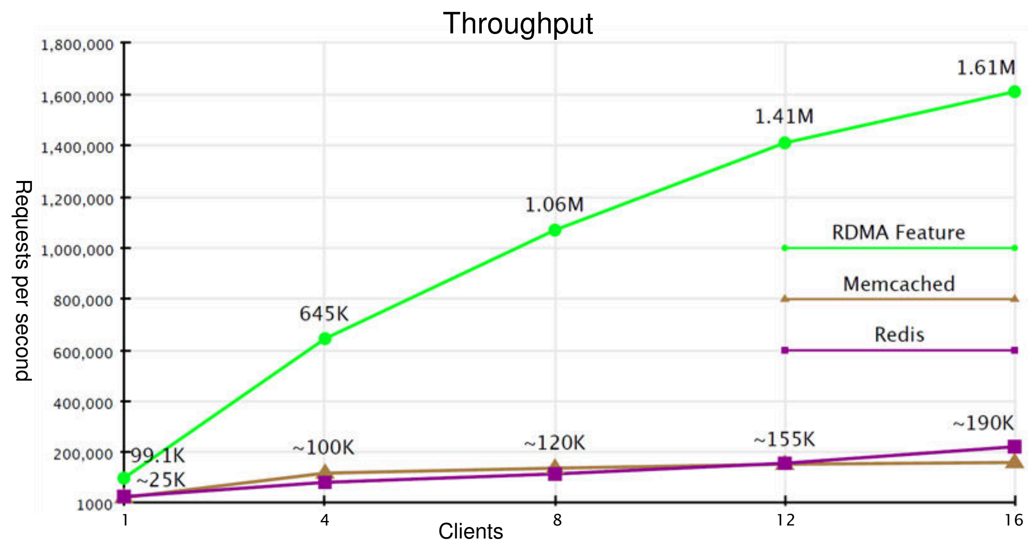
10X throughput increase with 90% latency reduction

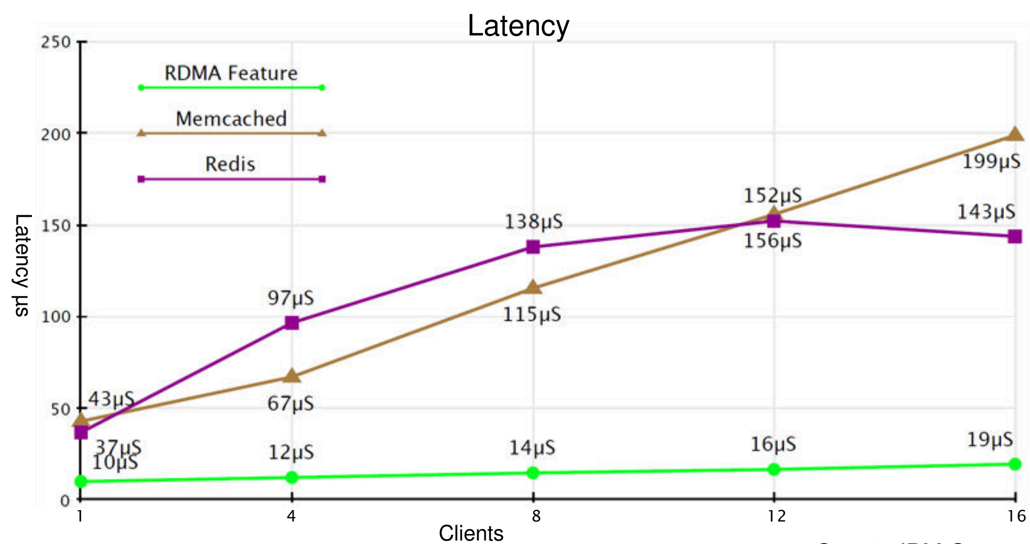
Experiment setup @ Lab environment

- Java™ clients
  - WXS over RoCE
  - Redis & Memcached over TCP
- CRUD workload
  - C10% R60% U20% D10%
- Payload 256B (w/ serialization)
- Single server; 1-16 clients
- 10Gb Ethernet w. RoCE

Experimental results

- CRUD throughput
  - WXS reaches **1.6 million** requests per second
  - WXS clients can drive **8.5X-10X more work** into the server
- CRUD latency
  - WXS latency **< 20µs (14µs avg.)**
  - **90% reduction**





Example: WXS RDMA scale-out

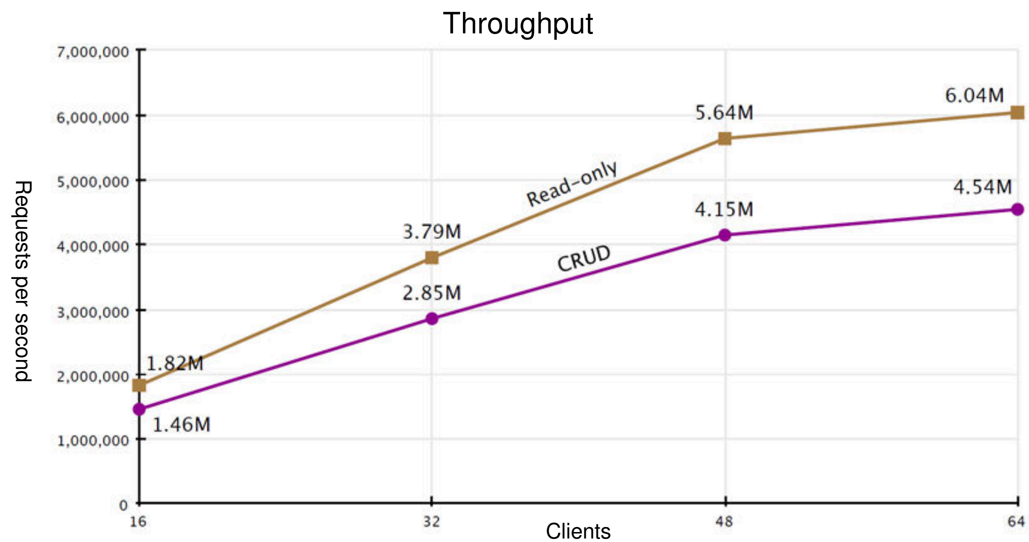
Near-linear scale-out to 6 million reads/s with a single server!

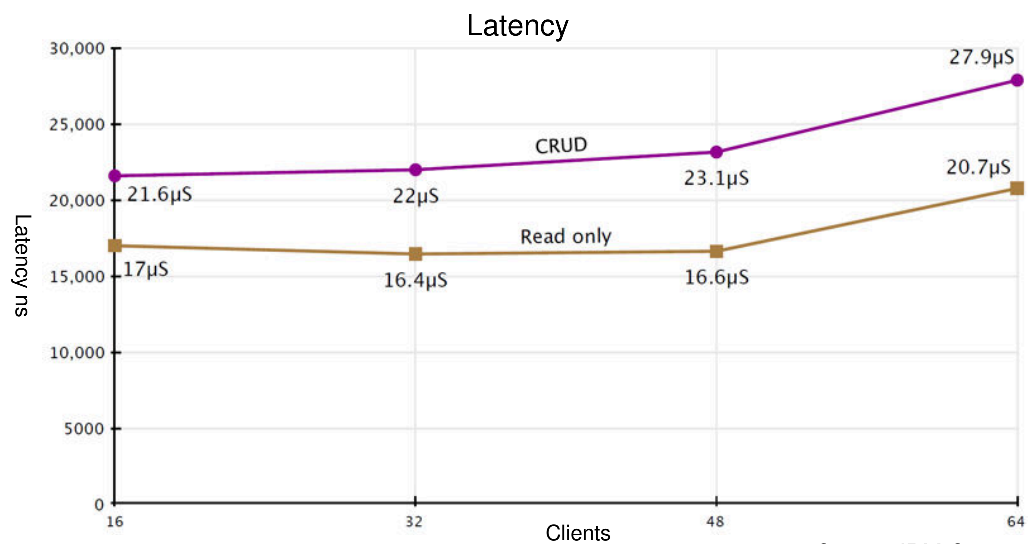
Experiment setup @ Lab environment

- Java client
  - WXS over IB
- CRUD workload
  - C10% R60% U20% D10%
- Payload 256B (w/ serialization)
- Single server; 16-64 clients
- 40Gb InfiniBand

Experimental results

- Throughput
  - CRUD**: 4.5 million** requests/s
  - Read-only: **6 million** requests/s
- Latency
  - CRUD**: 25µs avg.**
  - Read-only: **17µs avg.**





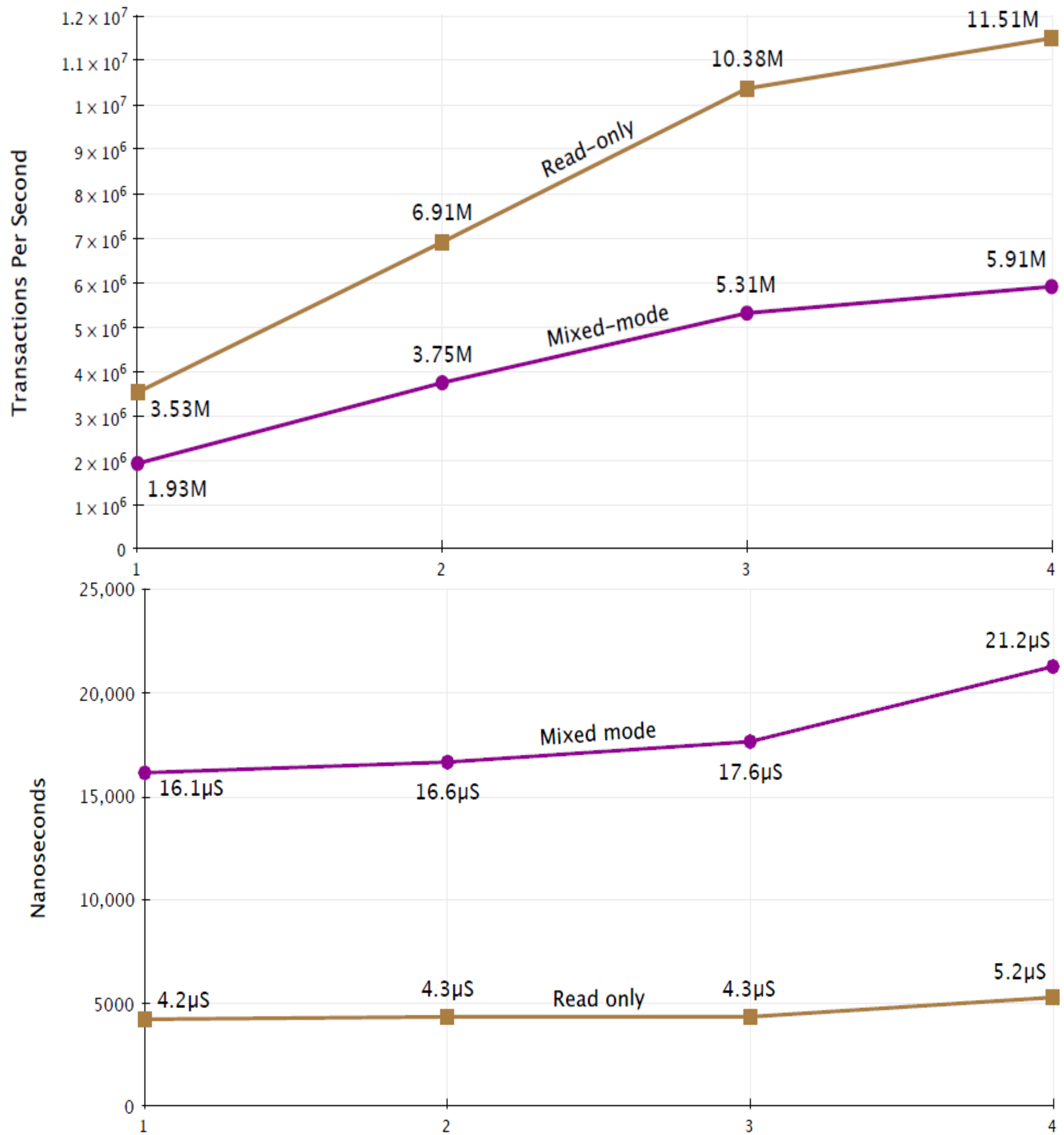
WXS RDMA scale-out: C client

C client

- CRUD workload
  - C10% R60% U20% D10%
- Payload 256B
- Single server; 16-64 clients
- 40Gb InfiniBand

Experimental results

- Throughput
  - CRUD: **5.9 million** requests/s
  - Read-only: **11.5 million** requests/s
- Latency
  - CRUD**: 17µs avg.**
  - Read-only: 4.3**µs avg.**



WXS RDMA - A True Game Changer

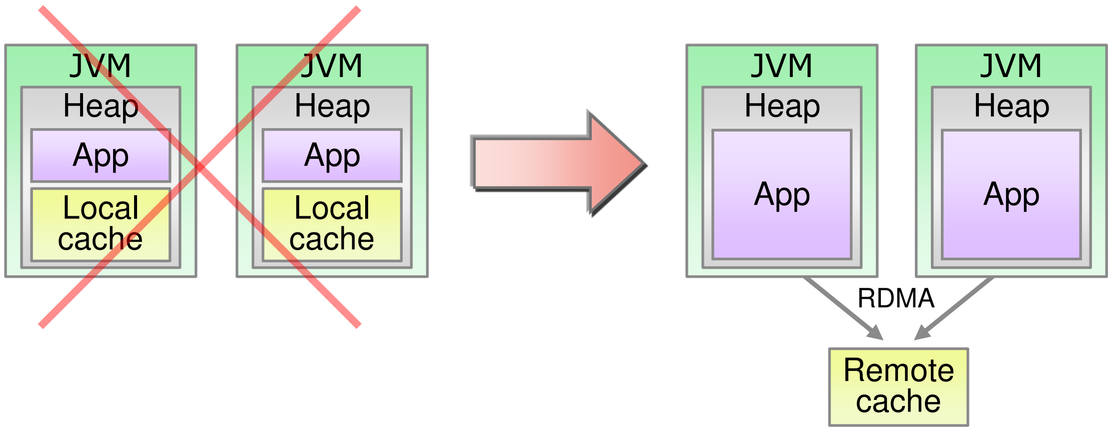
WXS RDMA Feature

- Addresses the **increasing scale-out pressures**
  - Vast numbers of mobile users, Internet of Things, the end of CPU performance scaling
- Enables a **new breed of scale-out systems**
  - Break the scale-out barriers with near-local access speed for remote data
- Allows enterprises to
  - Envision new, **game-changing applications** that take advantage of ultra-fast shared state and memory
  - Do more with less: **low carbon footprint** for high-velocity & high-volume caching applications

WXS RDMA scenarios

- **Focus on DynaCache scenarios** to improve Web application latencies
  - Content that is expensive to render or retrieve
  - Images, pages, page fragments, reference data, search results
- **Break the latency constraints** that force to hold the caches locally
  - RDMA's near-local access speed allows for remoting local data
- **Eliminate the secondary latencies** caused by local caching
  - Stop slowing down the applications by eating local JVM heap-memory
  - Stop wasting local CPU cycles either when each node renders the same content...or when the nodes replicate between one another

Example
- 2/3 of JVM's memory used for local cache (16GB)
- Cache duplicated across 48 JVMs

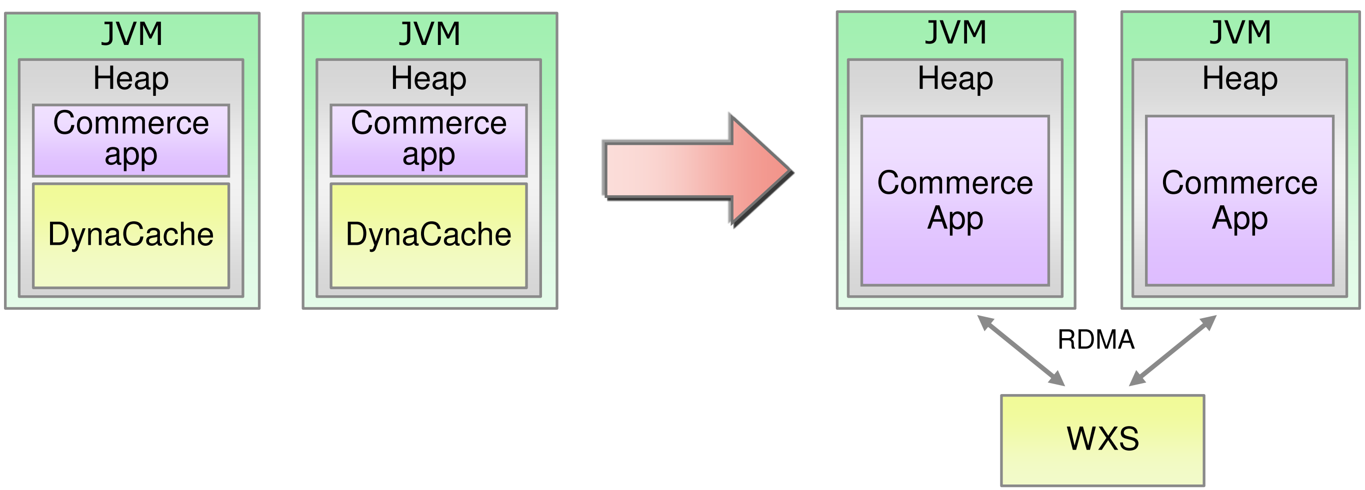


- Only one instance of the cache
- Increased performance from more memory and CPU for the apps
- Handle more web traffic with less hardware

Example: WXS RDMA & WebSphere Commerce

Faster response times with less hardware!

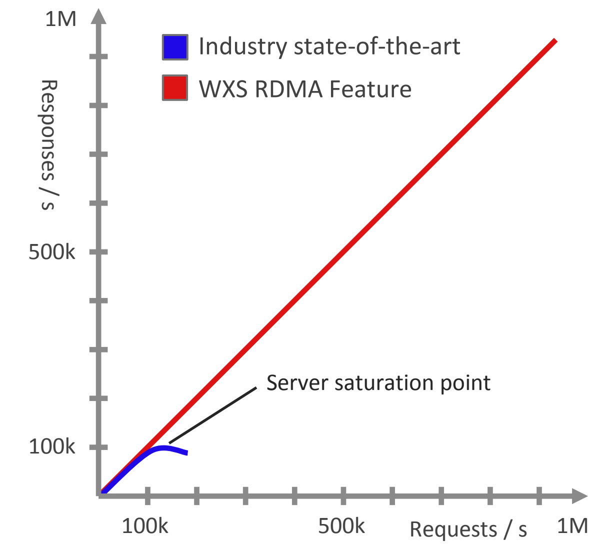
- **Experiment setup @ Lab environment**
  - WebSphere V7 64 bit, WebSphere Commerce V7 64 bit
  - Rendered products held in a local DynaCache vs. remote WXS
  - Cache size 4+ GB
  - 300-700 concurrent active users
- **Experimental results**
  - **40%- 50% end-user response time reduction** for 90th percentile (random product category browse)
  - Increased end-to-end application throughput



WXS and RDMA - Summary

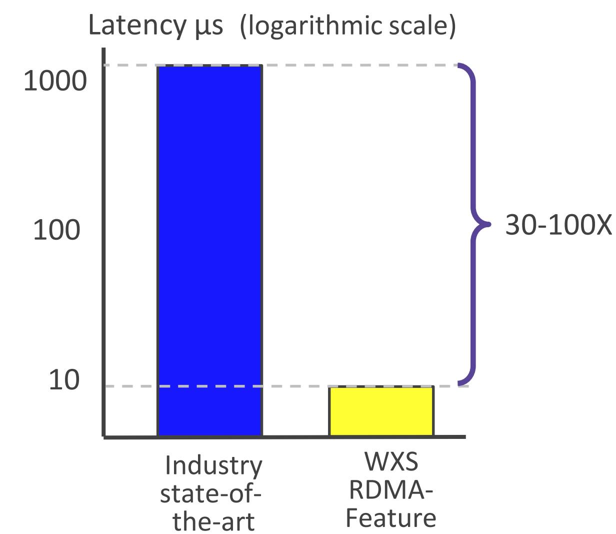
Ultra Performance - Caching Redefined!

- Exploring next gen distributed caching technologies architected from ground up to exploit RDMA
- Aim for near-local speed for remote access
  - Single digit microsecond read access latency (vs. industry state-of-the-art > 0.3 milliseconds)
- Scale up & out to Internet and PetaByte Scale
  - Target over a million requests per second throughput *per an individual server* (vs. industry state-of-the-art <200k/s)



Do more with less!

- Substantially reduce TCO and carbon footprint
- Accelerate latency-critical enterprise applications
  - ✓ Highly contextual and personalized applications
    - Commerce, Banking, Travel, Information services
  - ✓ Massive scale edge-caching scenarios
    - ISP, Mobile, Commerce, Portal
  - ✓ Internet Scale scenarios
    - Telco/Mobile, Internet of Things, Smarter City
  - ✓ Big Data & Instant Analytics
    - Commerce, Mobile, Banking, Credit Card



Outline

- Parallelization issue of data intensive enterprise java workloads on commercial multicore systems
- Design requirement of data centric computing systems
- Global secondary index for HBase (Hadoop Database)
- RDMA Enabled Cache
- Polyglot runtime systems & challenges for benchmarking community

Polyglot: the buzz

Java has ruled the enterprise application space for 15+ years but its dominance is cracking giving birth to …

Polyglot: No One Language Will Rule the Cloud

Modern Web apps simply refuse to be homogenized.

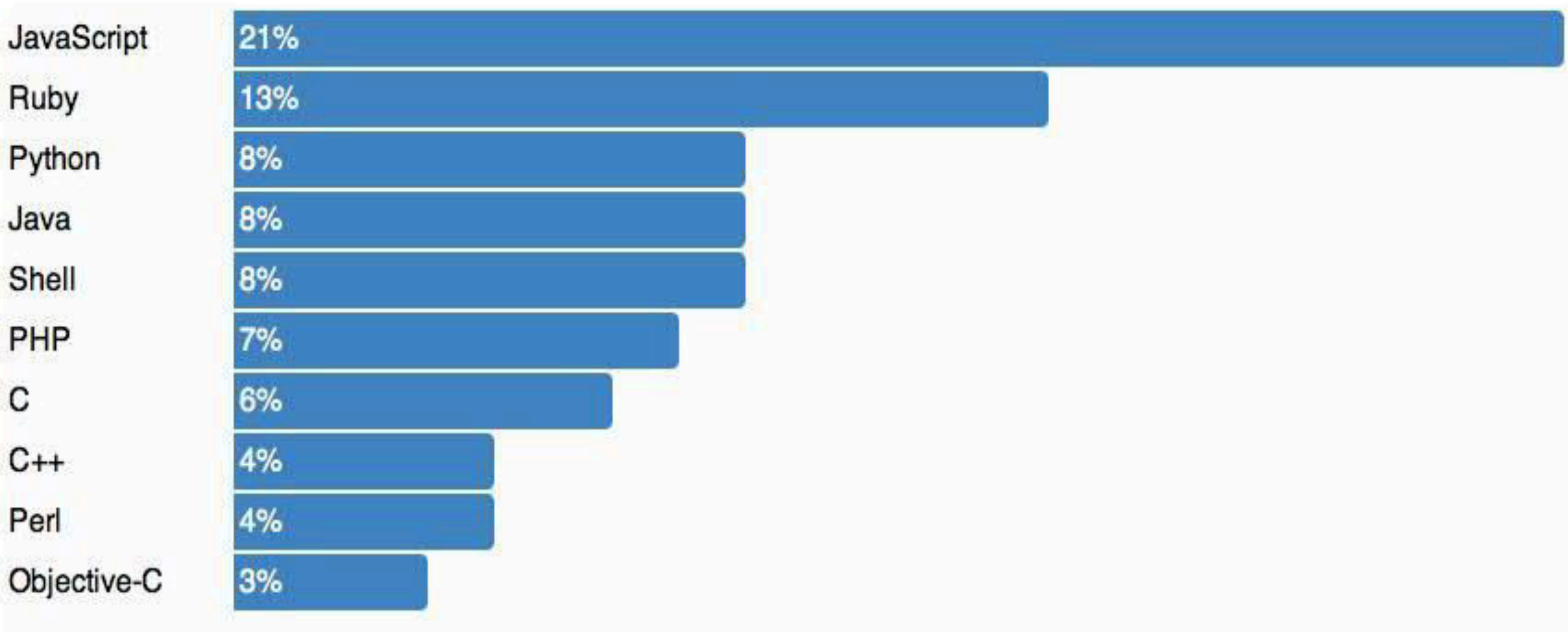
the reason you choose to go polyglot… is because you' re [always] choosing the best tool for the job

Businesses that already have the computing power are starting to consider scrapping their current application server architectures altogether, and to host their own Heroku-like platforms internally.

emergence of versatile new "polyglot programmers"

Future programmer: ~~java~~ a polyglot

Top Languages on Github (Cloud and Mobile)

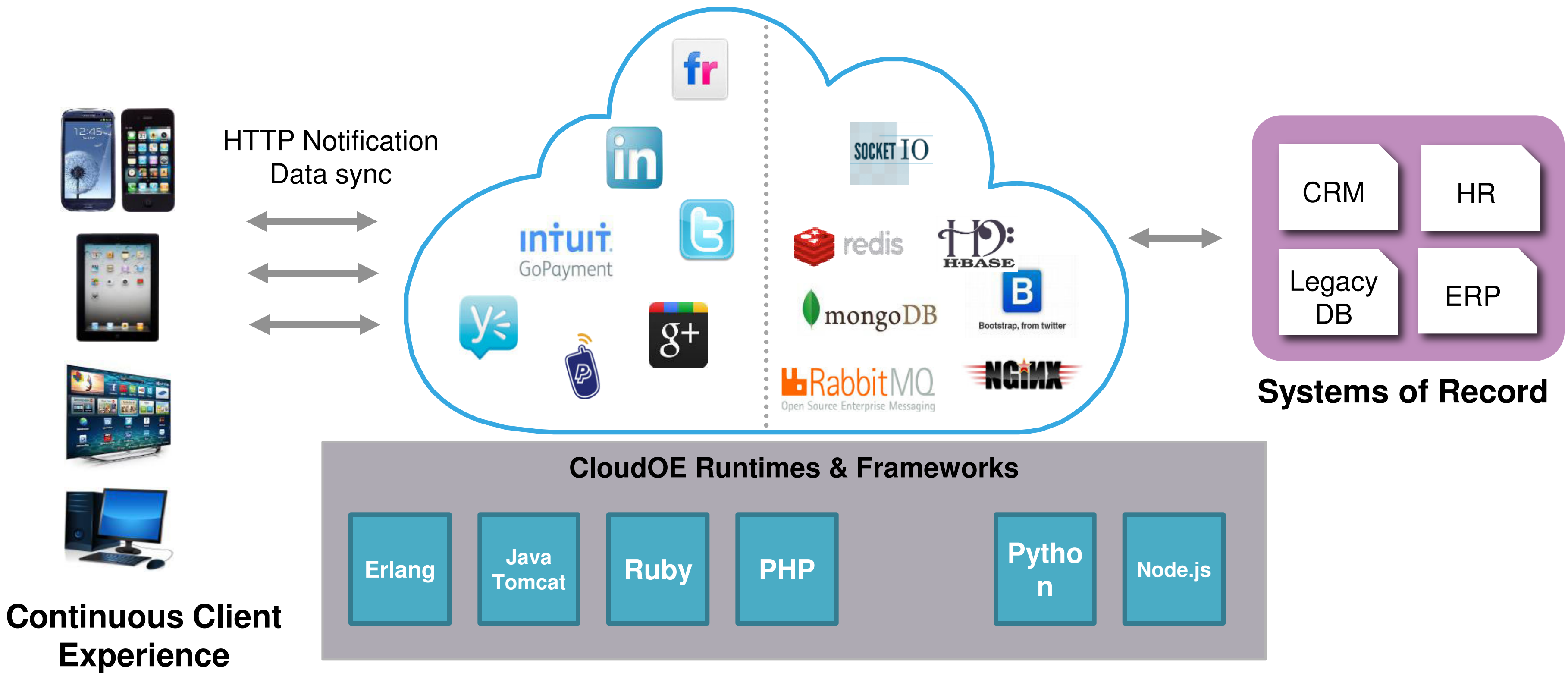


Web application projects in Github

JavaScript is by far the most popular

Future Application Platforms

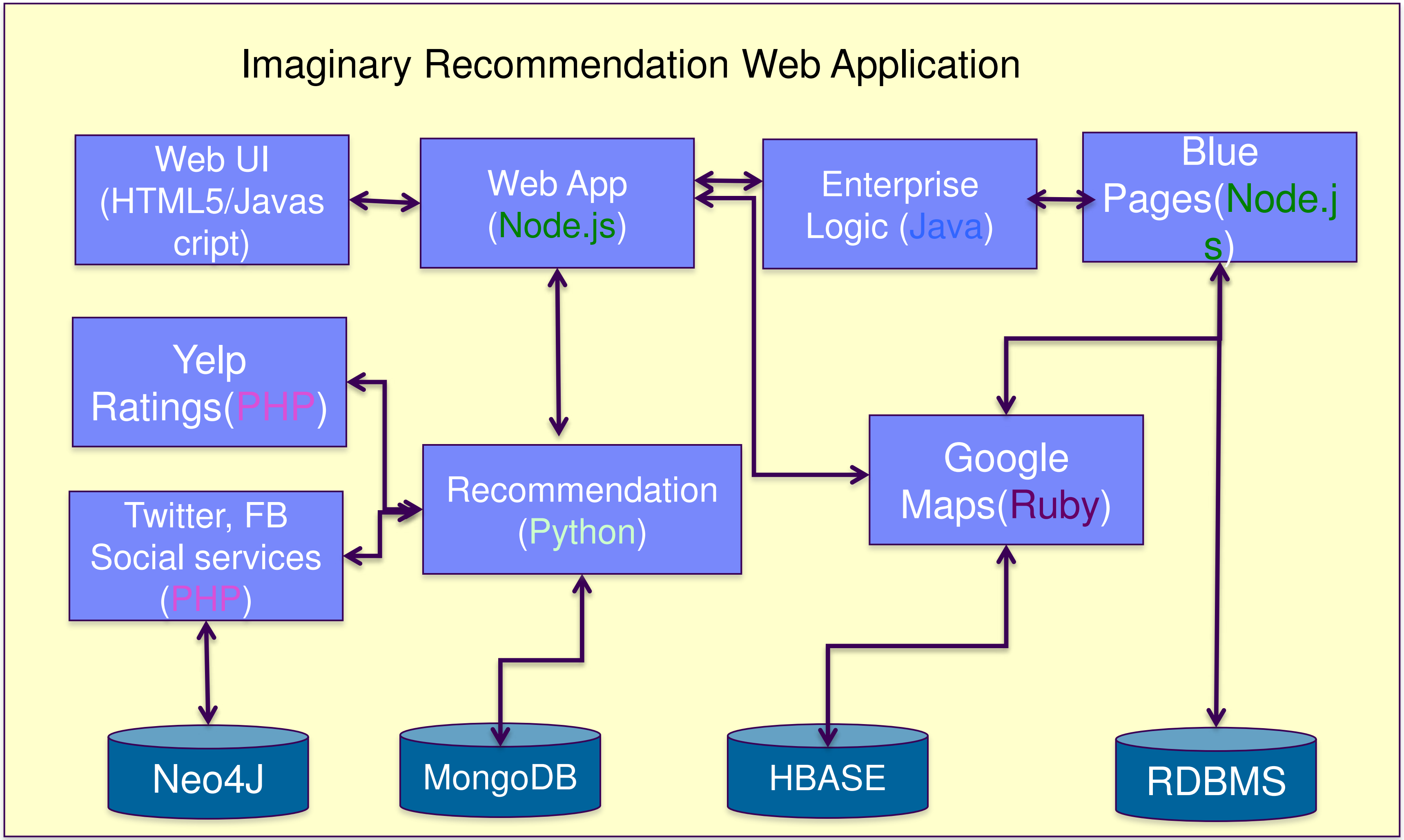
Cloud-centric applications require …



Future platforms are polyglot

Future Applications

What does a polyglot application looks like?



An Integrated Polyglot Platform

- Existing PaaS’s state-of-the-art
  - Excellent for simple web apps but anything beyond
  - Difficult to bring new services, provide QoS, enforce SLA, enable policy driven execution, provide visibility, traceability, governance, etc.
- Develop an elastic, scalable, language-independent **runtime platform** for **multi-language runtimes**
    - Provide common functions as services via the runtimes container
    - Design for elasticity, scalability and resilience
    - Provide standard interfaces to platform infrastructure components for scaling, logging, metering, deployment and optimizations
- Our research work will be the foundation for next generation cloud application platform: 1. initially targeted for Node.js applications, 2. with extensions for other languages/frameworks