COMP5349 – Cloud Computing

Week 12: Cloud Benchmarking

A/Prof Uwe Röhm
School of Information Technologies

Outline

- **Motivation**
- **Assessing Data Consistency Properties of Cloud Stores**
- **Yahoo Cloud Serving Benchmark - and YCSB+T**
- **HiBench for assessing cloud computing platforms**
- **Conclusions**

Benchmarking

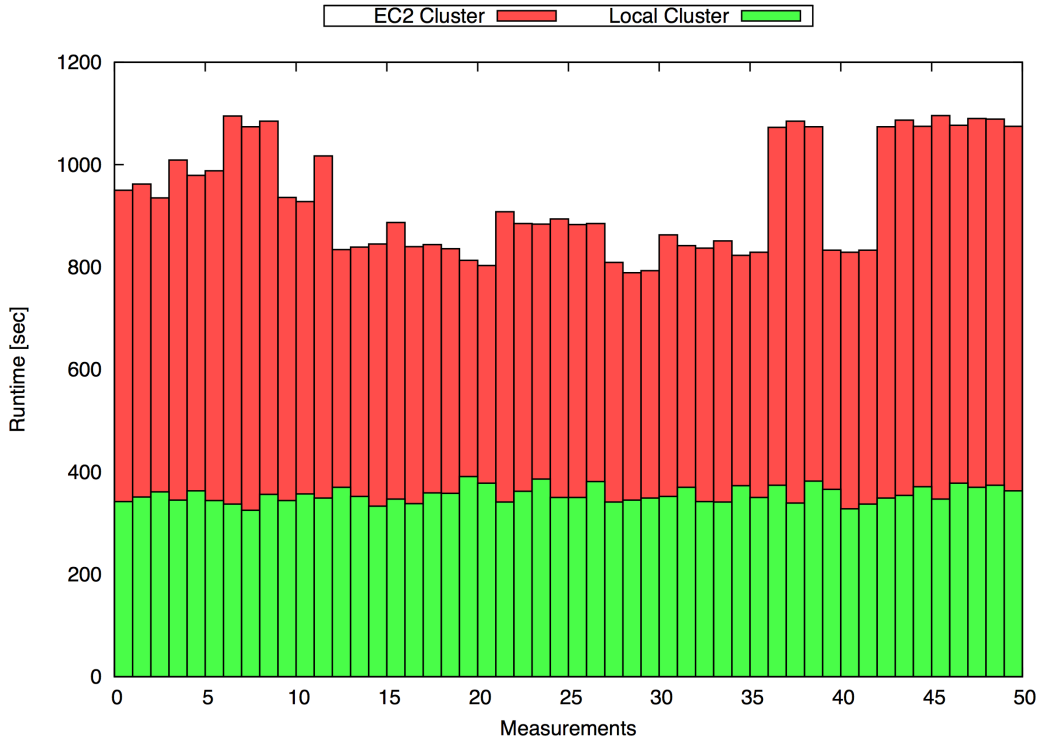
- Performance for cloud applications hard to estimate
- Cloud is 'black box' for outside
- Standard benchmarks (TPC-x or SPEC…) do not test the cloud metrics of interest
  - Such as elasticity or heterogeneity of hardware
  - Typically assume a static System-under-Test (SUT) for a controlled environment on which the benchmark is run with varying workload
  - TPC benchmarks model various specific transaction-processing scenarios with quite some emphasize on consistency requirements, but not the record-oriented usages of cloud services
- Hence several 'home-brew' approaches at the moment

Example 1: TPC-W Benchmark

[DBTest2009]

- TPC benchmark for web-based TP systems
- Models an online bookstore (as of 2002)
  - 14 defined web interactions (browse, search, display, order, …)
  - Different workload mixed of those: browsing, shopping, ordering
  - Tests full web stack: database up-to web server
  - Emulated browsers as workload
  - Test Metrics:
    - WIPS (web-interactions per second) for defined max. latency
    - Costs ($/WIPS)
- Issues in cloud settings:
  - Does not cater for scalability; in cloud setting, if latency becomes problem, we add machines => 'unlimited' WIPS
  - $/WIPS not clearly defined with the different cost models in the cloud
  - Fault tolerance and elasticity not measured

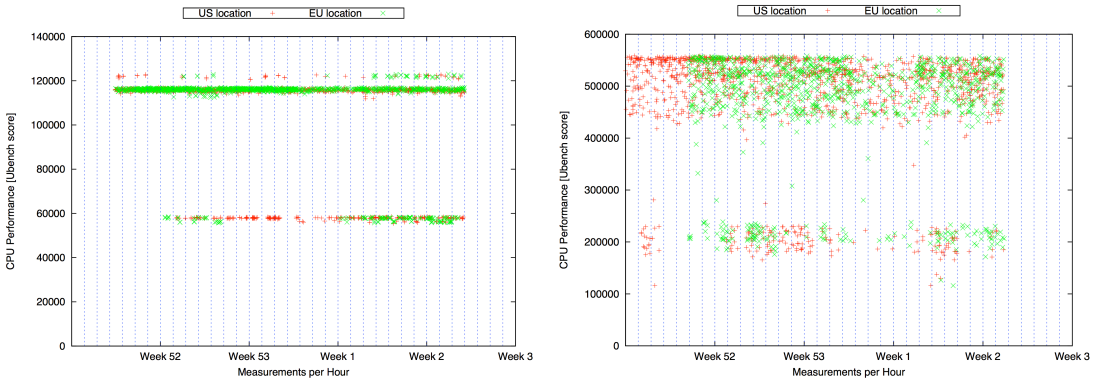
Example2: Cluster Performance Variation



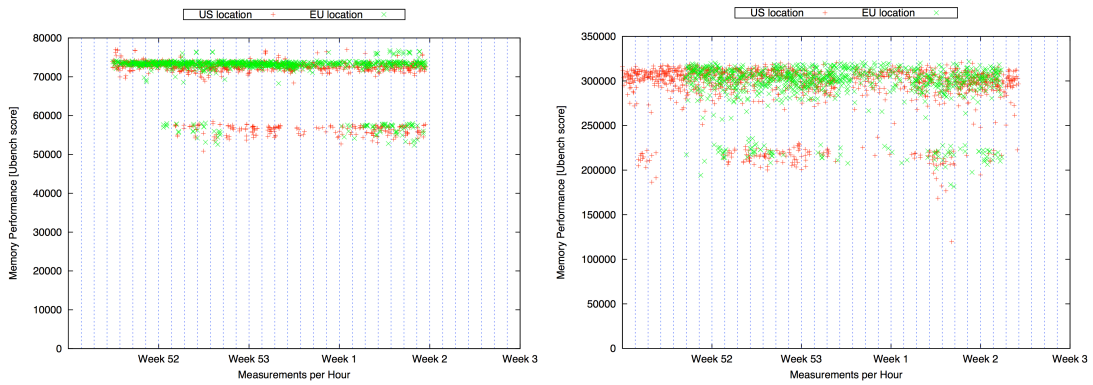
[Schad, VLDB2010]

EC2's CPU & Mem Variances (small and large instances)

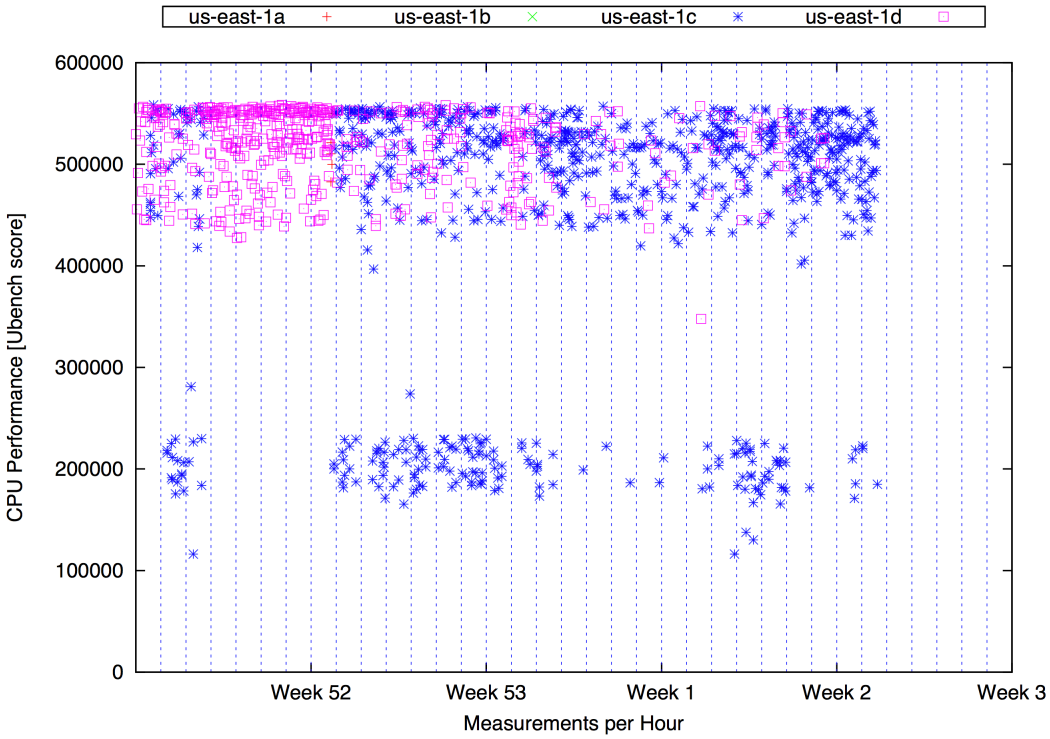
CPU:



Mem:

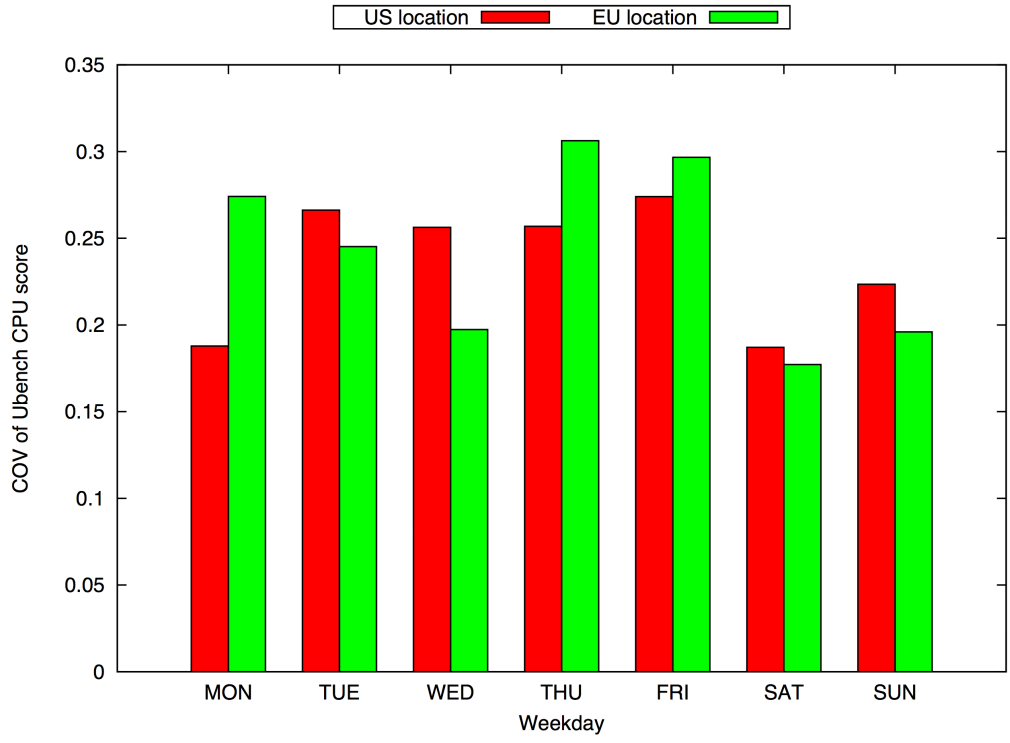


EC2 CPU Variability in US Region



[Schad, VLDB2010]

EC2's CPU Variability over time



[Schad, VLDB2010]

Cloud Performance Metrics

[YCSB - SoCC 2010]

- **Sizeup**
  - When is a given system saturated with requests?

  => keep hardware constant but increase workload; measure latency

- **Scale-out**
  - How does system perform as the number of machines increases?

  => increase data and number of machines, but keep workload same.
  Expect proportional scale in throughput, while latency remains same

- **Elasticity**
  - How does the system perform as number of machines increases while running?

    => Add machines to given system while running certain workload;
    expect performance improvement (how fast?) and no disruptions

- **Availability? Consistency?**

DATA CONSISTENCY PROPERTIES AND THE TRADE-OFFS IN COMMERCIAL CLOUD STORAGES: THE CONSUMERS' PERSPECTIVE

H. Wada, A. Fekete, L. Zhao, K. Lee and A. Liu.
NICTA and University of Sydney

CIDR 20111

The following slides are from this talk, available in the original from www.cidrdb.org/cidr2011/Talks/CIDR11_Wada.ppt

CIDR 2011: Consistency from the Consumer's Perspective

- Paper investigates consistency models provided by commercial cloud storages
  - If weak consistency, which extra properties supported?
  - How often and in what circumstances is inconsistency (stale values) observed?
  - Any differences between what is observed and what is announced from the vendor?

- Investigation of the benefits for consumer of accepting weaker consistency models
  - Are the benefits significant to justify consumers' effort?
  - When vendor offers choice of consistency model, how do they compare in practice?

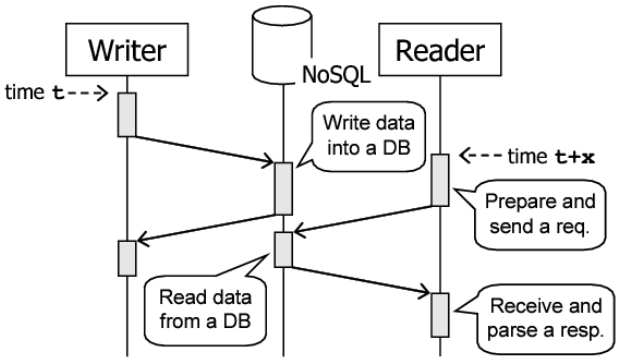
Observed Platforms

- A variety of commercial cloud NoSQLs that are offered as storage service
  - Amazon S3
    - Two options: Regular and Reduced redundancy (durability)
  - Amazon SimpleDB
    - Two options: Consistent Reads and Eventual Consistent Reads
  - Google App Engine datastore
    - Two options: Strong and Eventual Consistent Reads
  - Windows Azure Table and Blob
    - No option available in data consistency

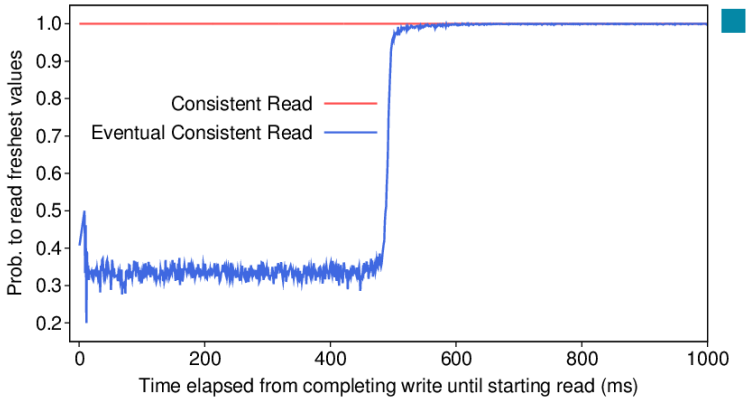
Frequency of Observing Stale Data

- Experimental Setup
  - A writer updates data once each 3 secs, for 5 mins
    - On GAE, it runs for 27 seconds
  - A reader(s) reads it 100 times/sec
    - Check if the data is stale by comparing value seen to the most recent value written
    - Plot against time since most recent write occurred

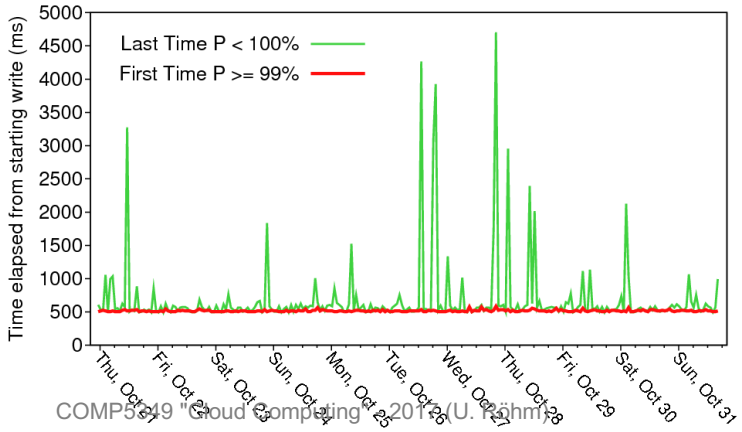
– Execute the above once every hour
  - On GAE, every 10 min
  - For at least 10 days
  - Done in Oct and Nov, 2010



SimpleDB: Read and Write from a Single Thread



- With eventual consistent read, 33% of chance to read freshest values within 500ms
  - Perhaps one master and two other replicas. Read takes value randomly from one of these?



- First time for eventual consistent read to reach 99% "fresh" is stable 500ms
- Outlier cases of stale read after 500ms, but no regular daily or weekly variation observed

Stale Data in Other Cloud Data Stores

Cloud NoSQL and Accessing Source	What Observed
SimpleDB (access from one thread, two threads, two processes, two VMs or two regions)	Accessing source has no affect on the observable consistency. Eventual consistent reads have 33% chance to see stale value, till 500ms after write.
S3 (with five access configurations)	No stale data was observed in ~4M reads/config. Providing better consistency than SLA describes.
GAE datastore (access from a single app or two apps)	Eventual consistent reads from different apps have very low ($3.3E^{-4}$ %) chance to observe values older than previous reads. Other reads never saw stale data.
Azure Storages (with five access configurations)	No stale data observed. Matches SLA described by the vendor (all reads are consistent).

Additional Properties: Read-Your-Writes?

- Read-your-writes: a read always sees a value at least as fresh as the latest write from the same thread/session
- Our experiment: When reader and writer share 1 thread, all reads should be fresh

- SimpleDB with eventual consistent read: does not have this property
- GAE with eventual consistent read: may have this property
  - No counterexample observed in ~3.7M reads over two weeks

Additional Properties: Monotonic Reads?

- Monotonic Reads: Each read sees a value at least as fresh as that seen in any previous read from the same thread/session
- Our experiment: check for a fresh read followed by a stale one
- SimpleDB: Monotonic read consistency is not supported
  - In staleness, two successive eventual consistent reads are almost independent
  - The correlation between staleness in two successive reads (up to 450ms after write) is 0.0218, which is very low

	2nd Stale	2nd Fresh
1st Stale	39.94% (~1.9M)	21.08% (~1.0M)
1st Fresh	23.36% (~1.1M)	15.63% (~0.7M)

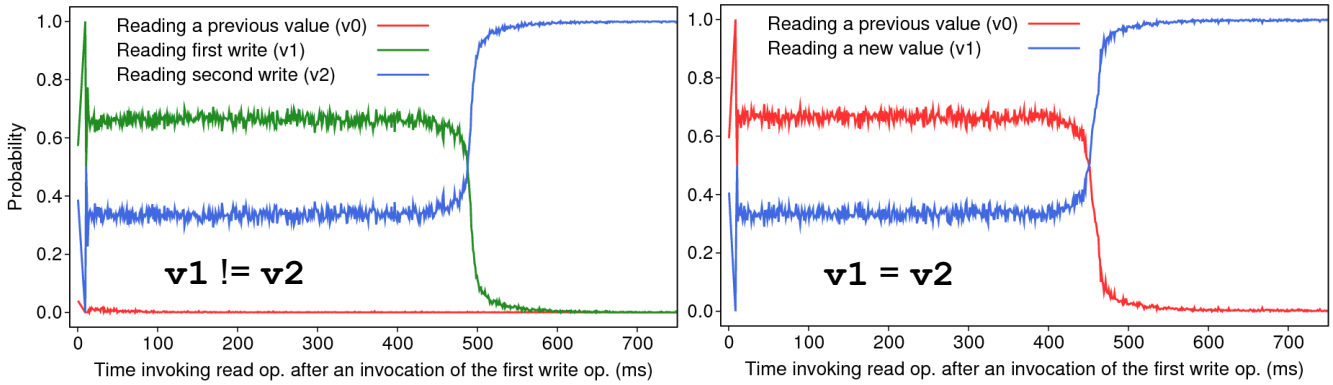
- GAE with eventual consistent read: not supported
  - $3.3E^{-4}$ % chance to read values older than previous reads

Additional Properties: Monotonic Writes?

- Monotonic Writes: Each write is completed in a replica after previous writes have been completed
- Programming is “notoriously hard” if monotonic write consistency is missing
  - W. Vogels. Eventually consistent. Commun. ACM, 52(1), 2009.
- This is an implementation property, not directly visible to consumer. But we explore what happens when we do successive writes, and try to read the data

SimpleDB's Eventual Consistent Read: Monotonic Write

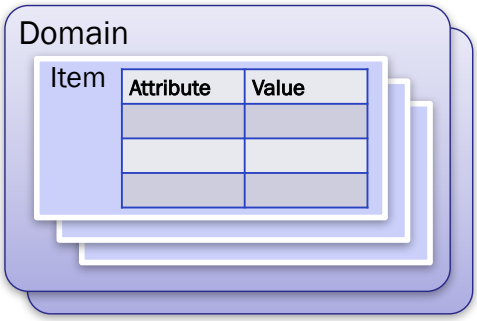
- A data has value **v0** before each run. Writing value **v1** and then **v2** there, then read it repeatedly



- When **v1** != **v2**, writing **v2** "pushes" **v1** to replicas immediately (previous value **v0** is not observed)
  - Very different from the "only writing one value" case
- When **v1** = **v2**, second write does not push (**v0** is observed)
  - Same as the "only writing one value" case

SimpleDB's Eventual Consistent Read: Further exploration -- Inter-Element Consistency

SimpleDB's Data Model



SimpleDB's API

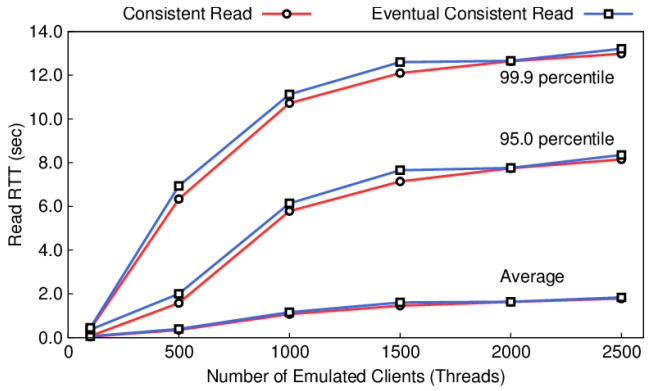
- Write a value
- Write multiple values in an item
- Write multiple values across items in a domain

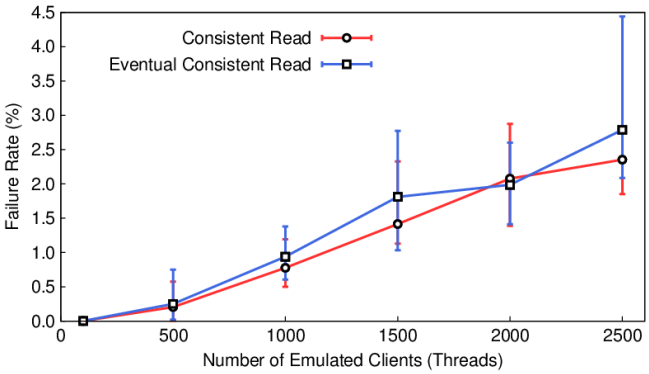
- Read a value
- Read multiple values in an item
- Read multiple values across items in a domain

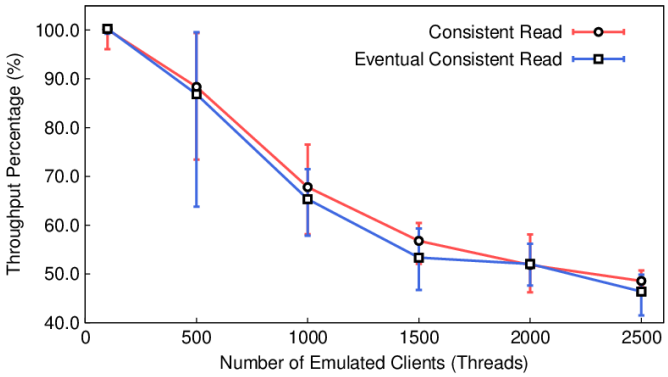
- Consistency between two values when writing and reading them through various combinations of APIs

Reading two values independently	Each read has 33% chance of freshness. Each read operation is independent
Writing two at once and reading two at once	Both are stale or both are fresh. Seems "batch write" and "batch read" access to one replica
Writing two in the same domain independently	The second write "pushes" the value of the first write (but only if two values are different)

Trade-Off Analysis of SimpleDB: A Benefit for Consumer from Weak Consistency?







- No significant difference was observed in RTT, throughput, failure rate under various read-write ratios
  - If anything, it favors consistent read!
- Financial cost is exactly same

Each client sends 1 rps. All obtained under 99:1 read-write ratio

What Consumers Can Observe (as of the state of these experiments)

- SimpleDB platform showed frequent inconsistency
- It offers option for consistent read. No extra costs for the consumer were observed from our experiments
  - At least under the scale of our experiments (few KB stored in a domain and ~2,500 rps)

?? Maybe the consumer should always program SimpleDB with consistent reads?

What Consumers Can Observe (cont'd) (as of the state of these experiments)

- Some platforms gave (mostly) better consistency than they promise
  - Consistency almost always (5-nines or better)
  - Perhaps consistency violated only when network or node failure occurs during execution of the operation

?? Maybe the chance of seeing stale data is so rare on these platforms that it need not be considered in programming?

  - There are other, more frequent, sources of data corruption such as data entry errors
  - The manual processes that fix these may also be used for rare errors from stale data

Implications of these Experiments for Consumers?

- Can a consumer rely on our findings in decision-making? NO!
  - Vendors might change any aspect of implementation (including presence of properties) without notice to consumers.
    - e.g., Shift from replication in a data center to geographical distribution
  - Vendors might find a way to pass on to consumers the savings from eventual consistent reads (compared to consistent ones)
- The lesson is that clear SLAs are needed, that clarify the properties that consumers can expect

BENCHMARKING CLOUD SERVING SYSTEMS WITH YCSB

Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan and Russell Sears
Yahoo! Research

SoCC 2010

Yahoo Cloud Serving Benchmark (YCSB)

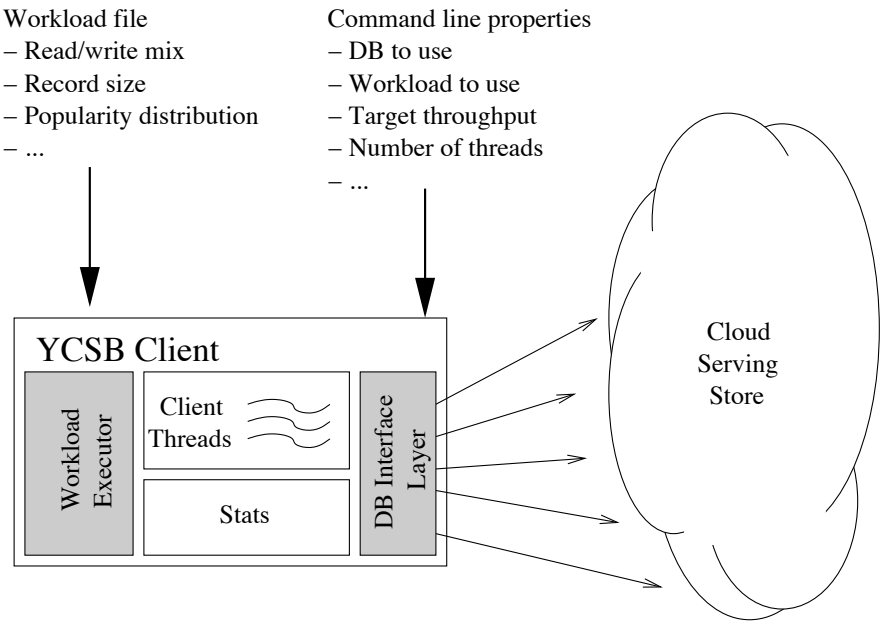


Figure 2: YCSB client architecture

Evaluation Setup

- Benchmarking four 'cloud' data stores
  - Cassandra 0.5.0 (key-value store base don DHT, similar to Dynamo)
  - HBase 0.20.3 (Apache's Hadoop database, similar to BigTable)
  - PNUTS (Yahoo's parallel cloud database, tables over DHTs)
  - Sharded MySQL 5.1.32 (client-side hashing)
- Hardware
  - Six node cluster (no cloud!)
    - Each node: dual quad-core 2.5GHz Xeon CPUs, 8 GB RAM, shared 6 disk RAID-10 array, Gigabit Ethernet
  - plus PNUTS on a 47 node cluster (not further specified)
  - YCSB client on a separate machine (8 core)
- Database
  - 120 million 1KB records (120 GB) => avg 20 GB of data per server

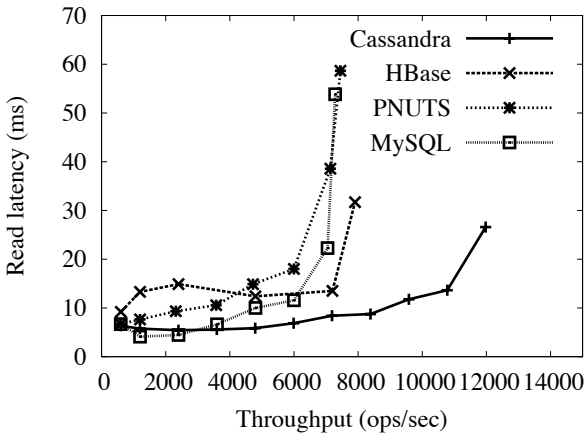
YCSB Workloads

Workload	Operations	Record selection	Application example
A—Update heavy	Read: 50% Update: 50%	Zipfian	Session store recording recent actions in a user session
B—Read heavy	Read: 95% Update: 5%	Zipfian	Photo tagging; add a tag is an update, but most operations are to read tags
C—Read only	Read: 100%	Zipfian	User profile cache, where profiles are constructed elsewhere (e.g., Hadoop)
D—Read latest	Read: 95% Insert: 5%	Latest	User status updates; people want to read the latest statuses
E—Short ranges	Scan: 95% Insert: 5%	Zipfian/Uniform*	Threaded conversations, where each scan is for the posts in a given thread (assumed to be clustered by thread id)

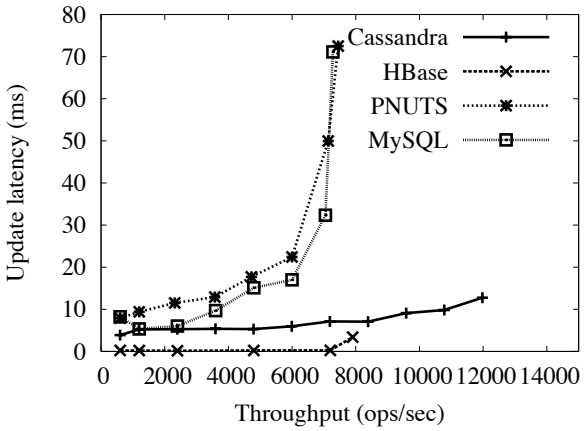
:load E uses the Zipfian distribution to choose the first key in the range, and the Uniform distribution to choose the number of re scan.

Sizeup of Different Systems for Workload A: Update Heavy (50:50)

Read Operations



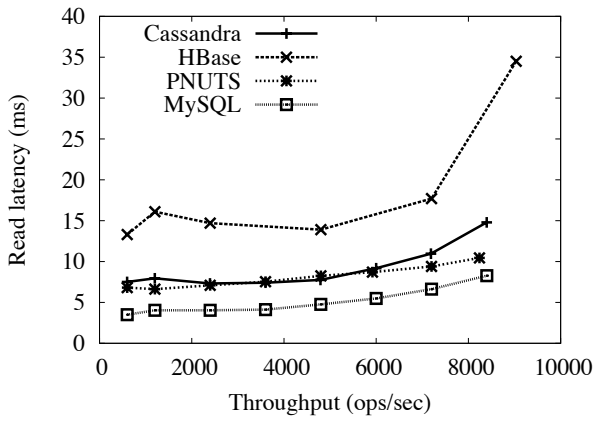
Write Operations



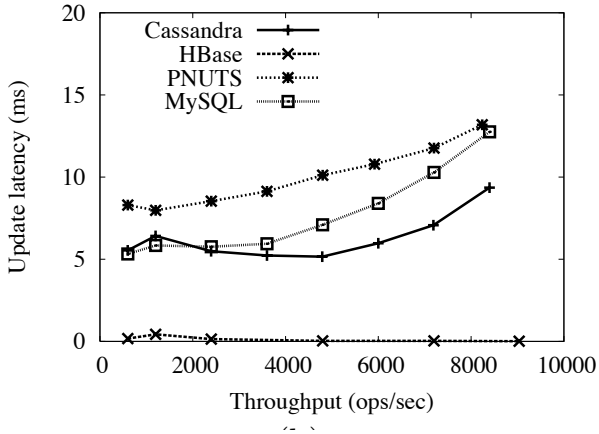
Note: PNUTS runs on top of own MySQL server

Sizeup of Different Systems for Workload B: Read Heavy (95:5)

Read Operations

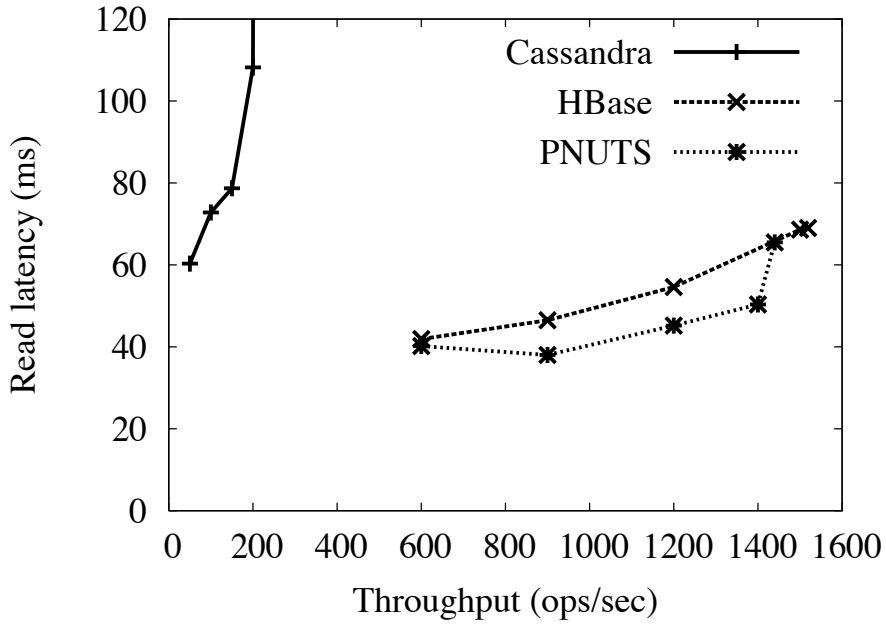


Write Operations



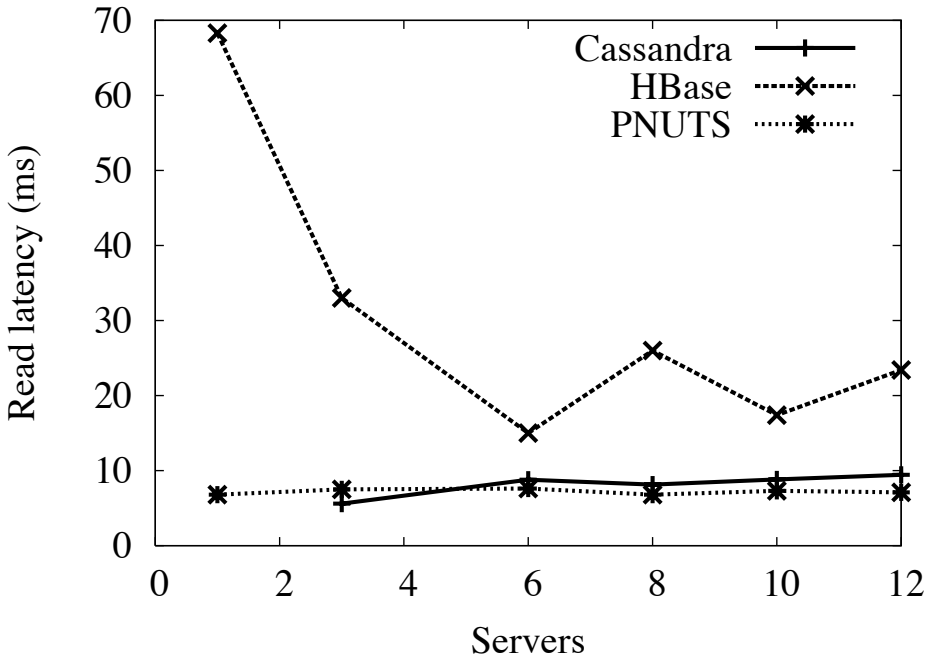
Both HBase and Cassandra construct records with multiple I/Os.
PNUTS runs on top of own MySQL server

Sizeup of Different Systems for Workload E: Short Scans



MySQL used hashing for sharing, hence no ranges possible;
Cassandra in 0.5.0 not optimized for scans either;
HBase gets much faster than PNUTS for larger scans

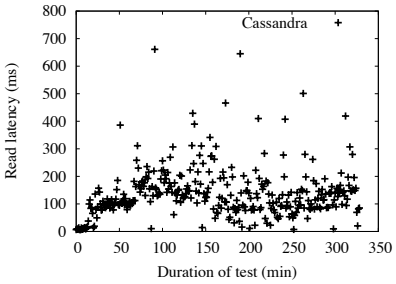
Scalability



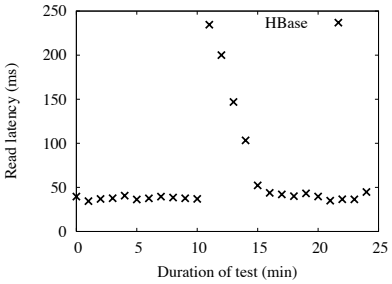
Read performance as cluster size increases
(and also varying data size and request rate correspondingly)
HBase behaviour is 'erratic' for <3 nodes

Elasticity

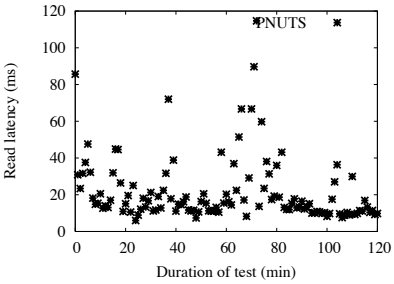
Time series showing impact of adding new server online:
Started with 5 servers, then added 6th server after 10 minutes.



(a) Cassandra



(b) HBase



(c) PNUTS

Cassandra needs hours to re-org data in DHT

HBase is faster to adapt as moves data later

PNUTS stabilises after ca. 80 mins

YCSB+T: YCSB+T: Benchmarking Web-scale Transactional Databases

[Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Roehm, IEEE CloudDB 2014]

- Based on standard YCSB
- In addition to existing two benchmark tiers of YCSB (Tier 1 (Performance) and Tier 2 (Scalability)), two more tiers:
  - Tier 5: Transactional Overhead
  - Tier 6: Consistency
- Transactional Overhead Tier
  - Quantifies overhead of database operations
    - Latency of database CRUD and scan operations at client
    - Latency of DB start(), abort() and commit()
    - in both transactional and non-transactional modes
- Consistency Tier
  - detect consistency anomalies in the data introduced during execution
  - quantify the amount of anomalies detected

HIBENCH SUITE

A BigData Microbenchmark suite
IEEE ICDE Workshops 2010

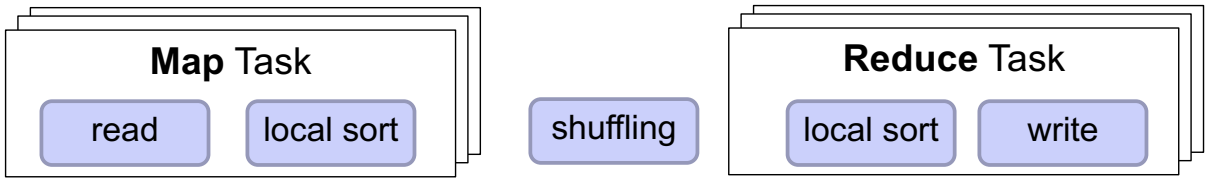
Intel / Apache's HiBench

https://github.com/intel-hadoop/HiBench

- Benchmark-Suite for cloud computing platforms
  - Collection of popular algorithms and workloads (online and offline analytics) as performance indicators:
  - Micro-benchmarks
    - Sort, WordCount, TeraSort
  - Web Search
    - Web Search Indexing, PageRank
  - Machine Learning
    - Bayesian Classification, *k*-Means Clustering
  - Analytical Queries
    - Scan, Join and Aggregation
  - Streaming
- v1.0 in 06.2012, current v6.0
  - superseded by BigDataBenchmark which includes this as subset?

Example Benchmark-Part: TeraSort

- Sorting is a popular benchmark workload, suitable for effective PR
  - cf. Jim Gray Sort Benchmark
  - cf. Hadoop Terabyte Sort Benchmark 2008
- Suitable for measuring pure performance of big data engines
  - No data transformation with user-defined logic
  - Basic facilities of engines used: read – shuffle/sort – output
  - Range-partitioning based on data sampling needed to guarantee total order



TeraSort with Spark and Flink

- TeraSort with Spark
  - Two RDDs (#partitions = #blocks of input file)

```
input = sc.textFile(inputPath)
partitioned = input.repartitionAndSortWithinPartitions(partitioner)
partitioned.saveAsTextFile(outputPath)
```

- TeraSort with Flink

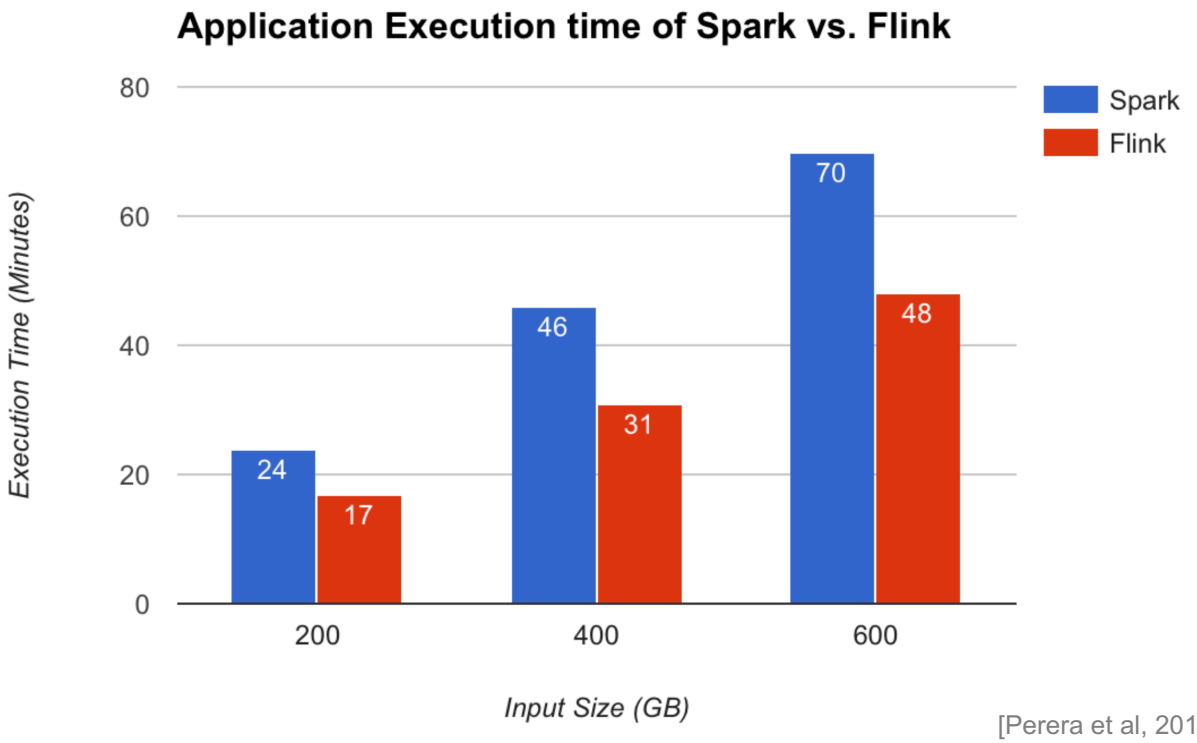
```
input = env.readTextFile(inputPath)
partitioned = input.partitionCustom(partitioner, 0)
sortedPartitioned = partitioned.sortPartition(0, Order.ASCENDING)
sortedPartitioned.output(outputPath)
```

HiBench Example Usage

Shelan Perera, Ashansa Perera, Kamal Hakimzadeh. *Reproducible Experiments for Comparing Apache Flink and Apache Spark on Public Clouds*. ArXiv 1610.04493, 2016.

- Cluster of EC2 instances
  - m3.xlarge for master nodes (4 vCPUs, 16 GB memory)
  - i2.4xlarge for worker nodes (16 vCPUs, 122 GB memory)
- Apache Spark 1.5.1 vs. Apache Flink 0.10.1
- Benchmarks:
  - TeraSort – i/o-bound global sorting benchmark
  - Stream Processing

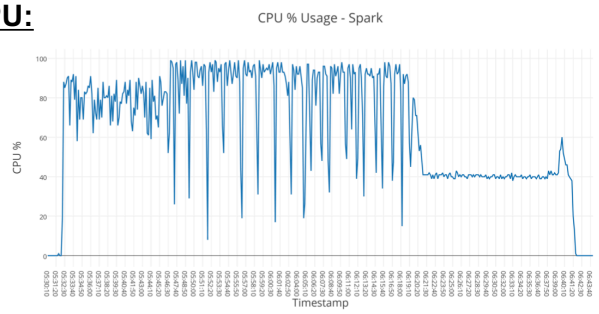
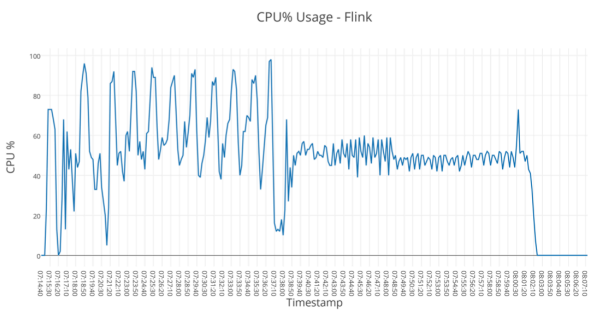
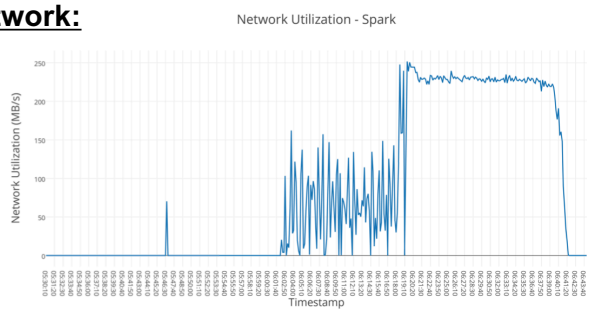
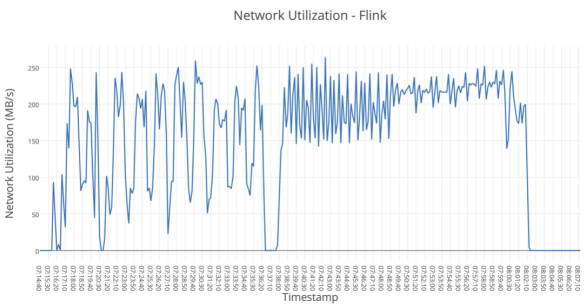
TeraSort Benchmark Results



[Perera et al, 2016]

TeraSort Benchmark Result Details

[Perera et al, 2016]

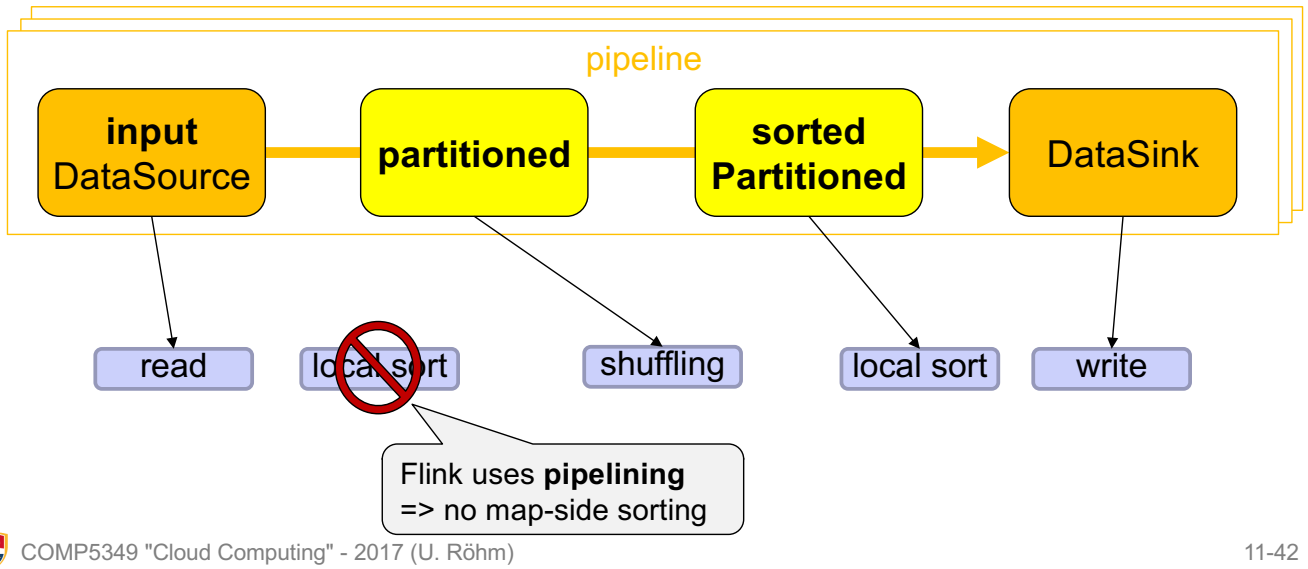
	Apache Spark	Apache Flink
CPU:		
Network:		

Disk: ***not shown (cf paper), but also used to fully assess any potential bottlenecks***

Shows Pipelining of Flink

- TeraSort with Flink:

```
input = env.readTextFile(inputPath)
partitioned = input.partitionCustom(partitioner, 0)
sortedPartitioned = partitioned.sortPartition(0, Order.ASCENDING)
sortedPartitioned.output(outputPath)
```



Big Data Benchmarking

- Lots of interest, but many challenges
  - data generation
  - scalability
  - workload variety
- various benchmarks suites available
- "Big Data Benchmarking" workshop series

Table 1. Comparison of data generation techniques in existing big data benchmarks.

Benchmark efforts	Volume	Velocity	Variety (data sources)	Veracity
Hibench [12]	Partially scalable	Un-controllable	Texts	Un-considered
GridMix [4]	Scalable	Un-controllable	Texts	Un-considered
PigMix [6]	Scalable	Un-controllable	Texts	Un-considered
YCSB [9]	Scalable	Un-controllable	Tables	Un-considered
Performance benchmark [15]	Scalable	Un-controllable	Tables, texts	Un-considered
TPC-DS [11]	Scalable	Semi-controllable	Tables	Partially Considered
BigBench [11]	Scalable	Semi-controllable	Texts, web logs tables	Partially Considered
LinkBench [17]	Partially scalable	Semi-controllable	Graphs	Partially Considered
CloudSuite [10]	Partially scalable	Semi-controllable	Texts, graphs, videos, tables	Partially Considered
BigDataBench [19]	Scalable	Semi-controllable	Texts, resumes, graphs, tables	Considered

[R. Han and X. Lu: *On Big Data Benchmarking*. BPOE, LNCS, 2014]

References

- Carsten Binnig, Donald Kossmann, Tim Kraska, Simon Loesing: *How is the Weather tomorrow? Towards a Benchmark for the Cloud*. DBTest Workshop, 2009.
- Jörg Schad, Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz: *Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance*. VLDB 2010.
- Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, Russell Sears: *Benchmarking Cloud Serving Systems with YCSB*. SoCC 2010.
- Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang: *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis*. In: ICDE Workshops, 2010. Source code: https://github.com/intel-hadoop/HiBench
- R. Han and X. Lu: *On Big Data Benchmarking*. BPOE, LNCS, 2014.
- S. Perera, A. Perera, K. Hakimzadeh. *Reproducible Experiments for Comparing Apache Flink and Apache Spark on Public Clouds*. ArXiv 1610.04493, 2016.
- H. Wada, A. Fekete, L. Zhao, K.. Lee and A. Liu: *Data Consistency Properties and the Trade-offs in Commercial Cloud Storages: Consumers' Perspective*. CIDR2011.
- Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Roehm: *YCSB+T: Benchmarking Web-Scale Transactional Databases*. CloudDB 2014.