ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

An Evaluation of Alternative Architectures for Transaction Processing in the Cloud

Donald Kossmann, **Tim Kraska***, Simon Loesing

* now at UC Berkeley

Systems Group @ ETH Zurich

Systems@ETH Zürich

- Joint initiative of 4 professors, 6 PostDocs, ≈30 PhD students
- Mission: Redefining the „systems stack“ according to new app requirements and technology trends
  - Increasing scale, concurrency/parallelism at all levels, heterogeneity (HW, SW & data), diversity (ideas from other fields),...
- Research Approach
  - Build real systems (Barrelfish, Cloudy, Crescando...)
  - Collaboration within group (from HW to app)
  - Working closely with industry (e.g., ECC)
  - Rethinking and creating new ideas (e.g., merge DB & app server)

Cloud Computing Today

- The Cloud Computing era promises
  - Scalability
  - Fault-tolerance
  - Pay-as-you-go
- All big players and more and more startups offer cloud products
- Wide range of services offered
  - Infrastructure services (e.g., data storage, virtual machines...)
  - Platform services (e.g., Web-application hosting, map-reduce...)
- Resulting in a jungle of services
  - Divergent properties and guarantees
  - Different internal architectures
  - Often not compatible
  - ...

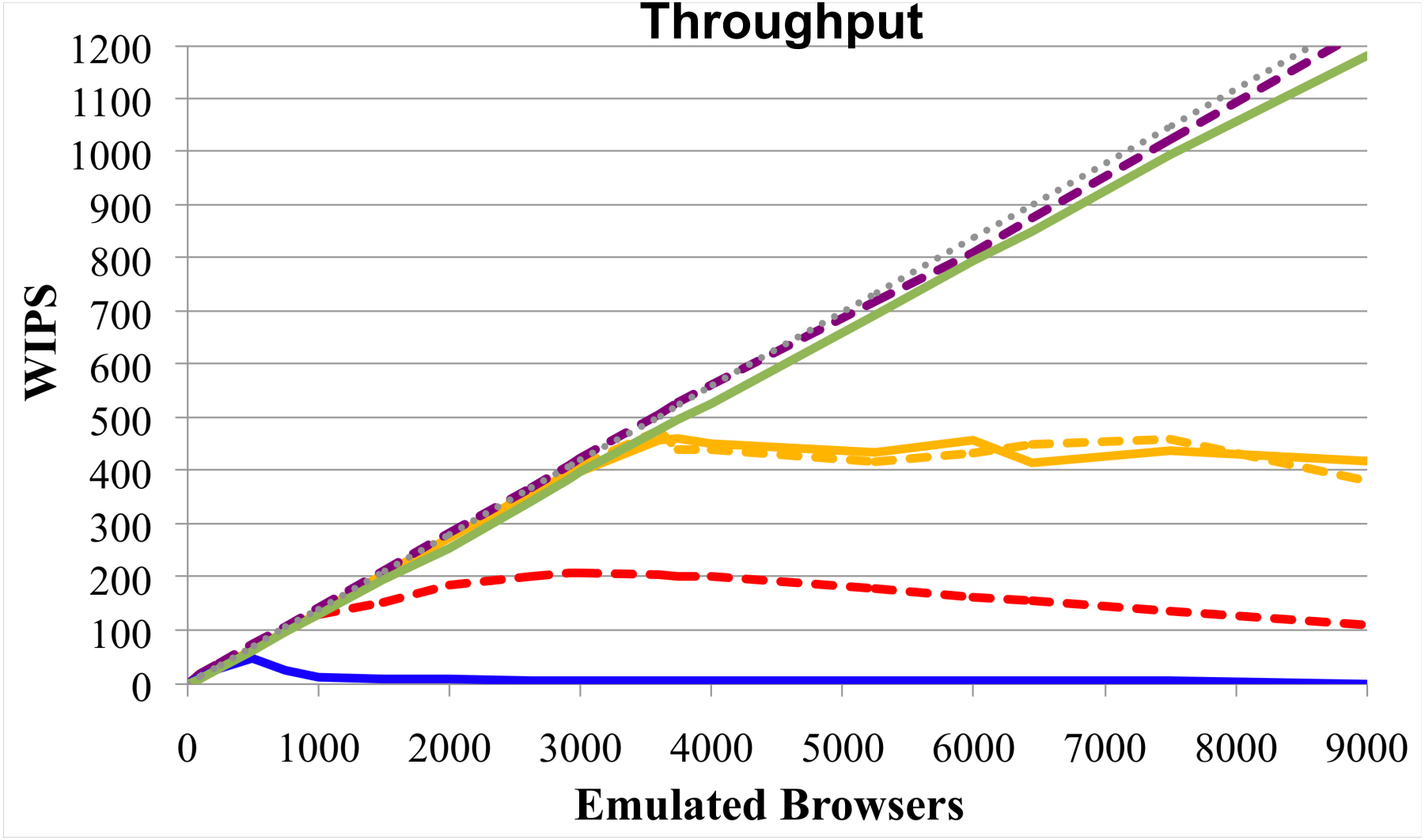
Lost in the Cloud?

Motivation

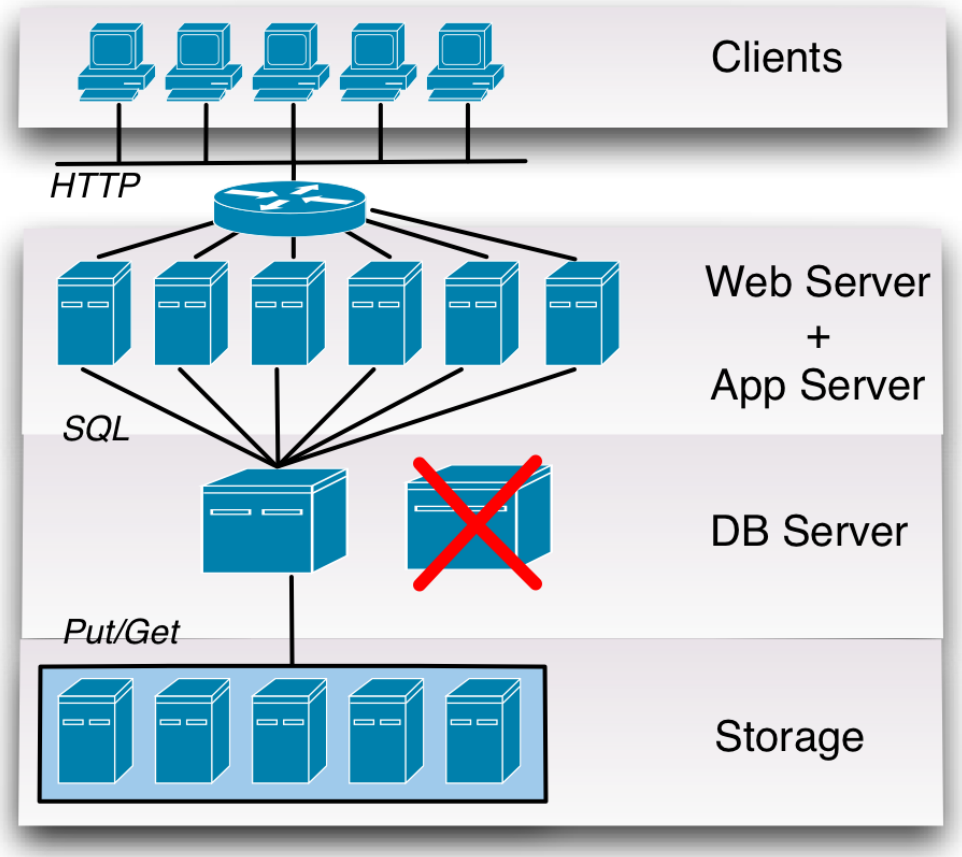
- We are trying to bring light into the dust!

- Questions:
  - **What are the building blocks to fulfill these promises?**
  - **Do current offerings really fulfill the cloud promises?**
  - **How do offerings compare in terms of scalability and cost?**

A Peek Ahead...


Throughput
WIPS
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Emulated Browsers

Traditional Database Architecture



- The Database-Tier limits application scalability!

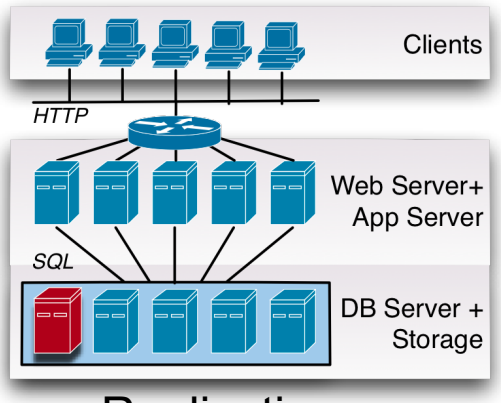
[R. Buck-Emden. *The SAP R/3 System. Addison-Wesley, 2nd edition, 1999*]

Related Work: Distributed Databases

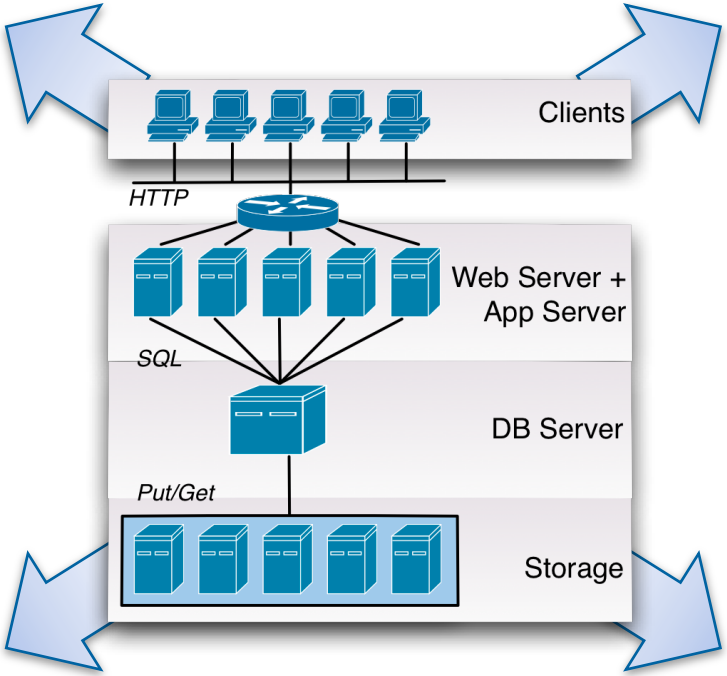
- Database is distributed accross multiple physical locations
- Common techniques:
  - Partitioning
    - Large number of possible partitioning schemes [1]
    - Repartitioning is very expensive
  - Replication
    - Replicating data increases fault-tolerance and read performance
    - Replication needs a mechanism to keep replicas consistent [2]

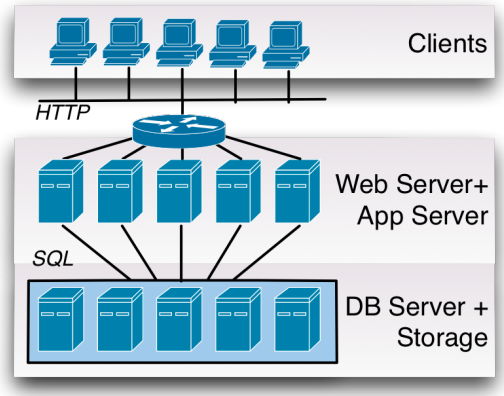
[1] S. Ceri and G. Pelagatti. Distributed databases principles and systems. 1984.
[2] P.Bernstein, et al. Concurrency Control and Recovery in Database Systems. 1987.

Cloud Architectures

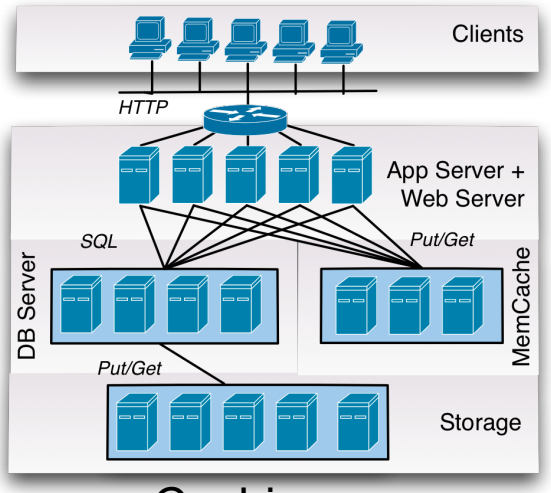


Replication

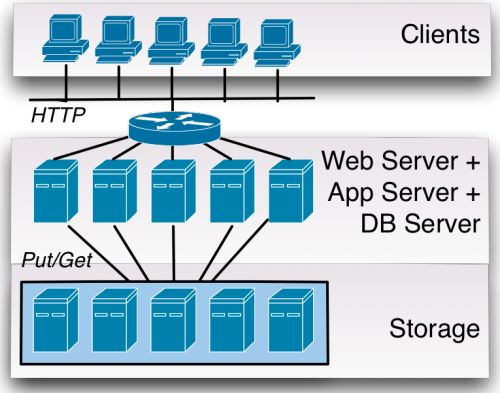




Partitioning



Caching



Distributed Control

Tested Services

	MS Azure	Google App Eng	AWS RDS	AWS S3
Business Model	PaaS	PaaS	IaaS	IaaS
Architecture	**Replication**	**Part. + Repl. (+Dist. Control)**	**Classic**	**Distr. Control**
Consistency	**SI**	**≈ SI**	**Rep. Read**	**EC**
Cloud Provider	Microsoft	Google	Amazon	Flexible
Web/App Server	.Net Azure	AppEngine	Tomcat	Tomcat
Database	SQL Azure	DataStore	MySQL	--
Storage / FS	Simple DataStore	GFS	--	S3
App-Language	C#	Java/AppEngine	Java	Java
DB-Language	SQL	GQL	SQL	Low-Lev. API
HW Config.	Part. automatic	Automatic	Manual	Manual

Tested Services

	AWS SimpleDB	AWS MySQL	AWS MySQL/R
Business Model	IaaS	IaaS	IaaS
Architecture	**Partitioning + Replication**	**Classic**	**Replication**
Consistency	**EC**	**Rep. Read**	**Rep. Read**
Cloud Provider	Amazon	Flexible	Flexible
Web/App Server	Tomcat	Tomcat	Tomcat
Database	SimpleDB	MySQL	MySQL
Storage / File System	--	EBS	EBS
App-Language	Java	Java	Java
DB-Language	SimpleDB Queries	SQL	SQL
HW Config	Partly Automatic	Manual	Manual

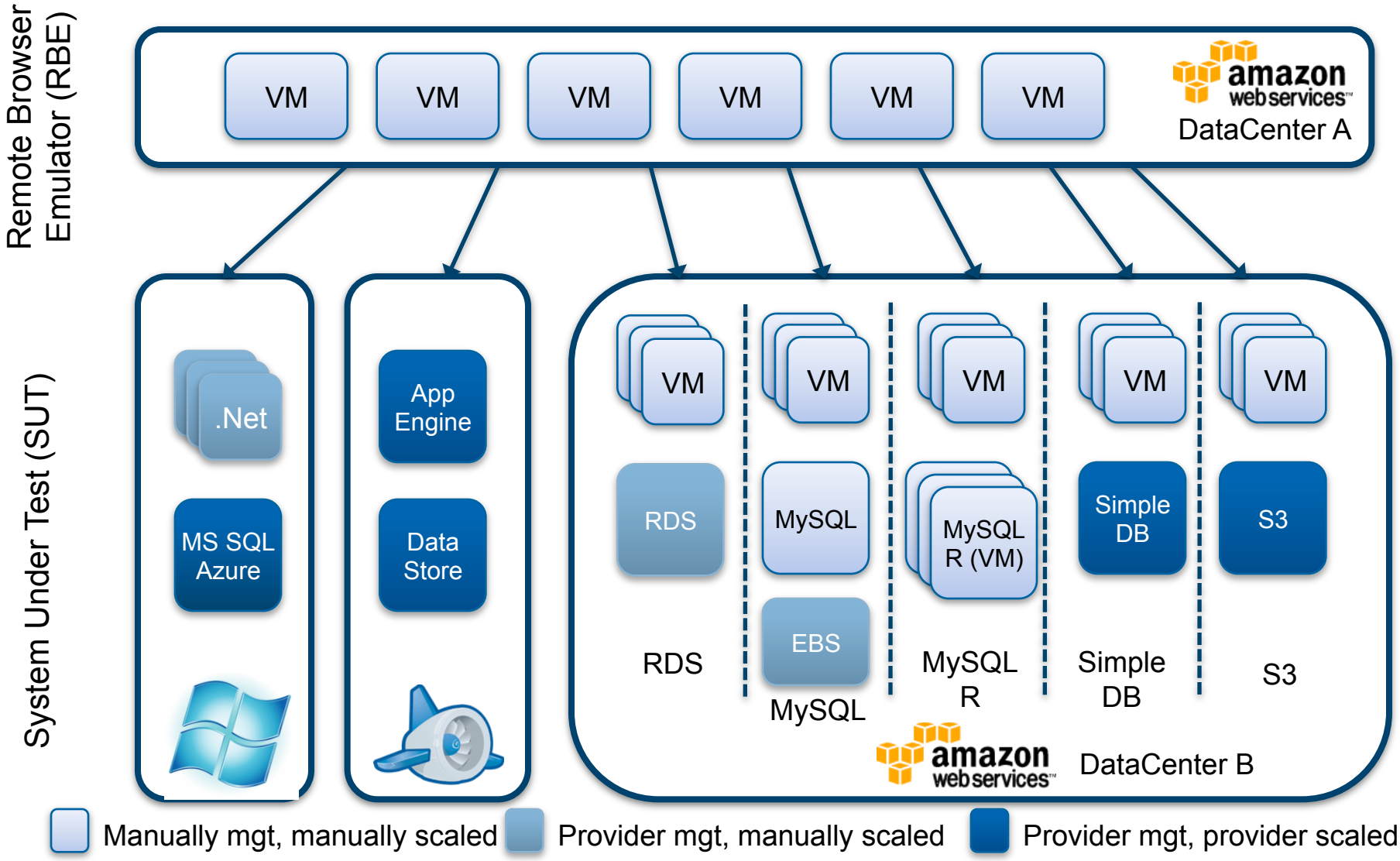
Trade-Offs you have to make

- **Consistency** — eventual consistency vs. ACID
- **Query language support** — key/value vs. SQL light vs. full SQL
- **Scaling (web/app-tier and db-tier)** — automatic vs. manual
- **Patches/Maintenance (web/app-tier and db-tier)** — automatic vs. manual
- **Language flexibility** — pre-defined vs. fully flexible
- **Machine access** — none vs. partial vs. full
- **Price-plans** — free quotas, fixed costs,…
- **Data Models** — key/value, semi-structured, relational
- **…**

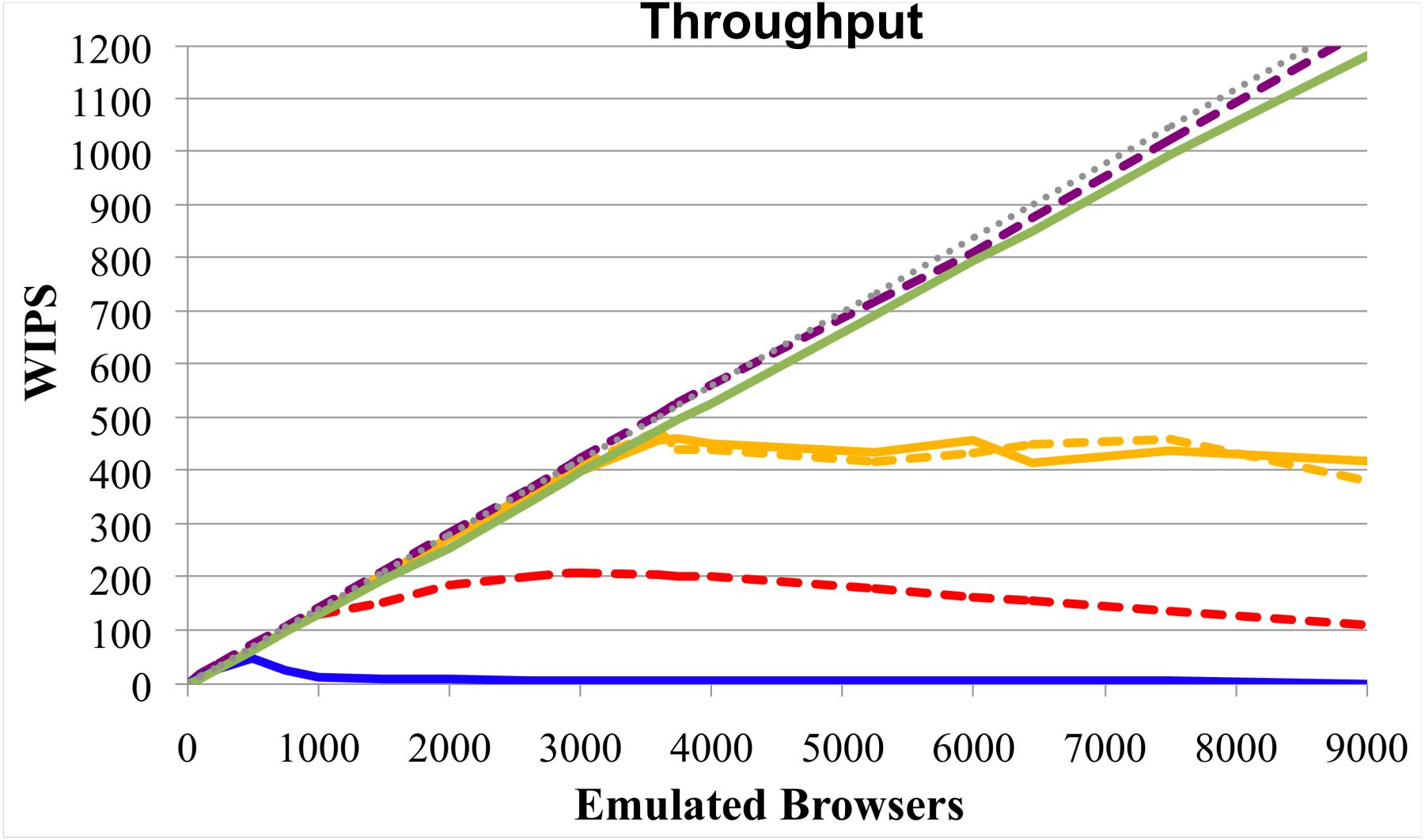
Benchmarking Cloud Services

- Goal: Do the services fullfill their promises?
- Benchmarking approach
  - TPC-W Benchmark, Ordering Mix
  - Adapted to fit the cloud requirements [DBTest 2009 paper]
- New main metrics:
  - **Throughput**
    - Max Web Interactions Per Second (WIPS) in response time
    - Increase load over time from 1 to 9000EB (1EB every 0.4sec)
    - 1 EB ~ 500 requests per hour, 9000EB ~1250 requests per second
  - **Cost**
    - Cost / WI ($): Divide costs of the running system by the current load
    - CostPerDay ($): The (projected) total cost of running the benchmark 24 hours
  - **Cost predictability**
    - s($) – Cost standard deviation from 1 EB to 9000EB

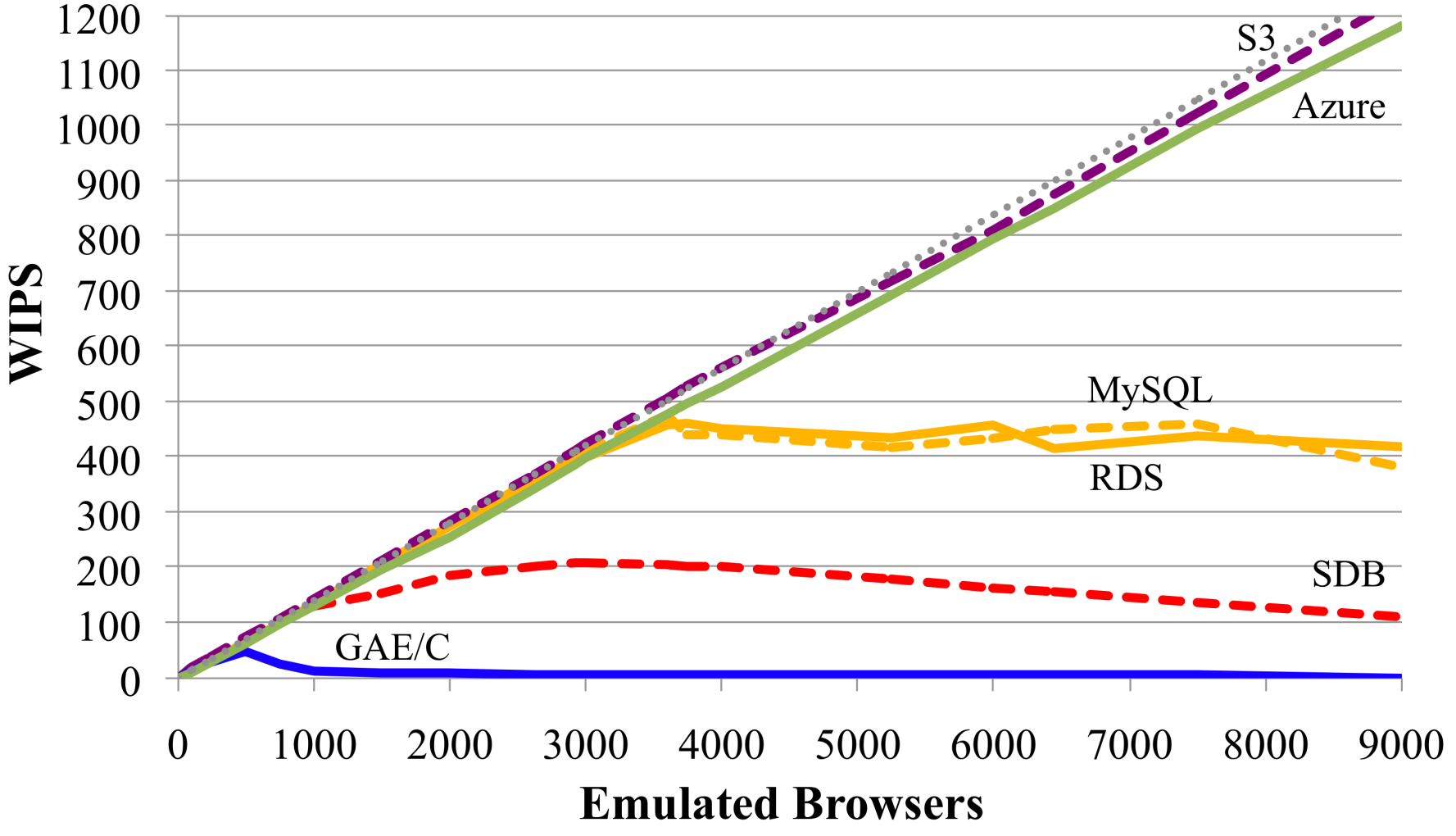
Benchmark Setup



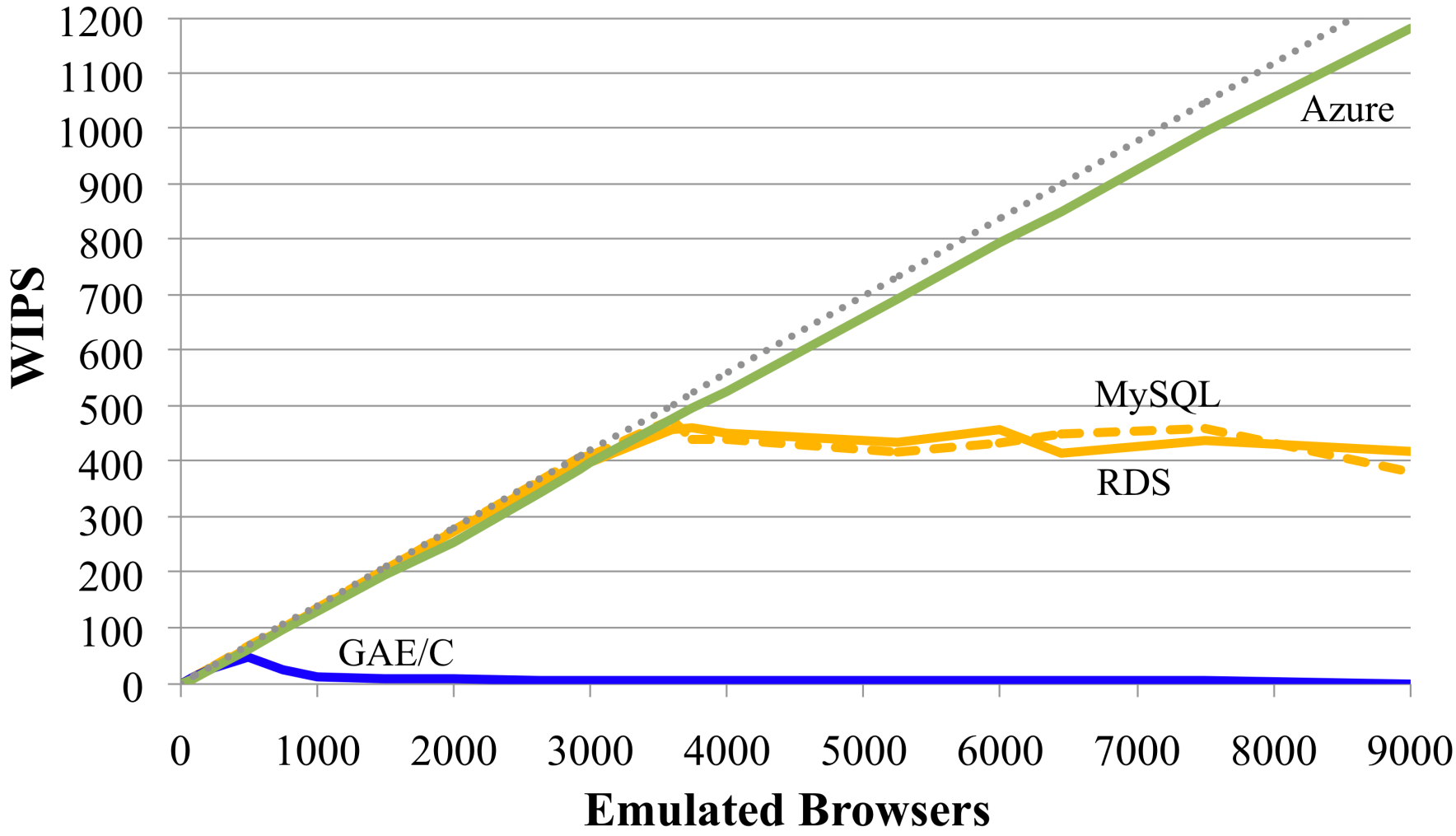
A Peek Ahead...


Throughput
WIPS
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Emulated Browsers

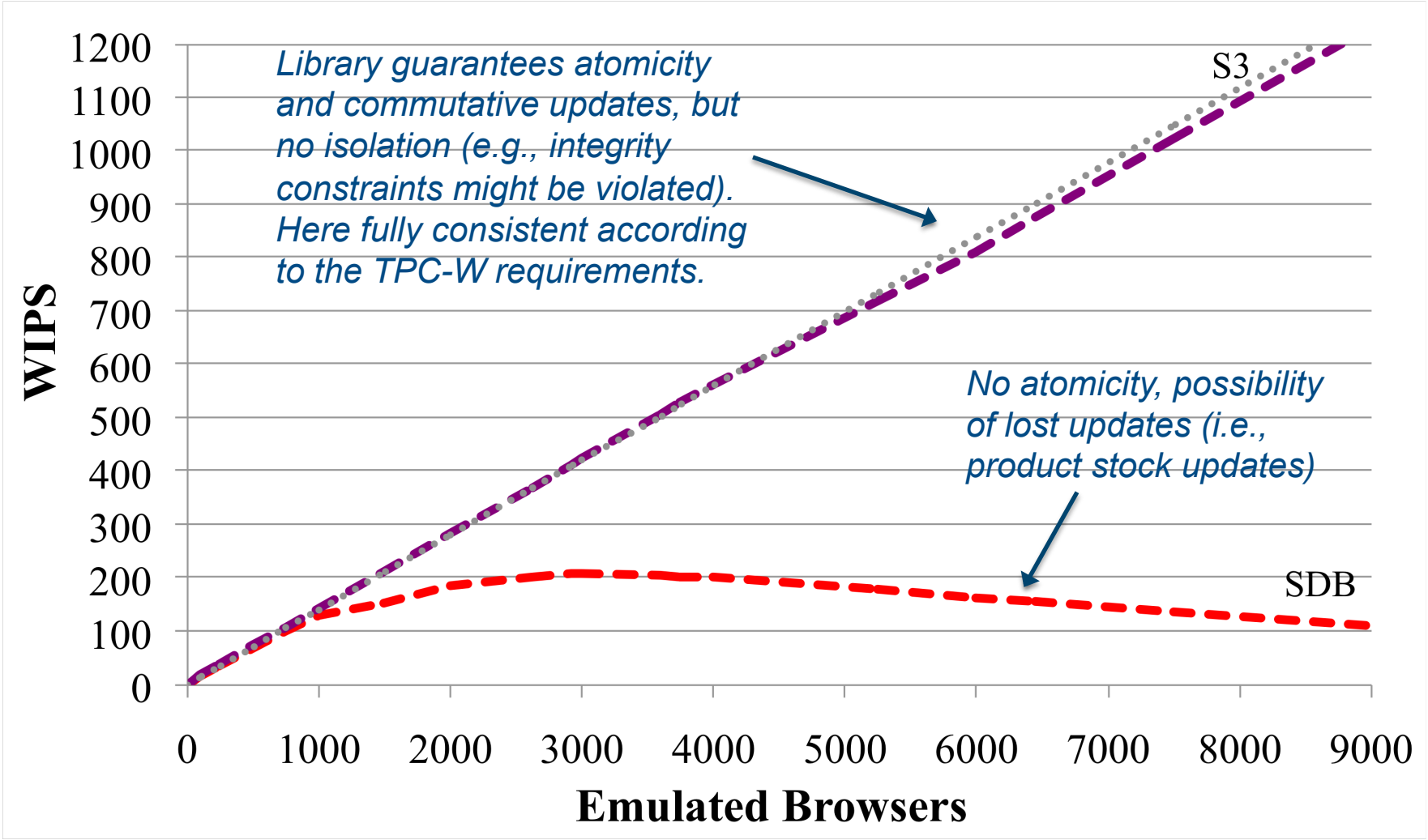
Throughput [WIPS]



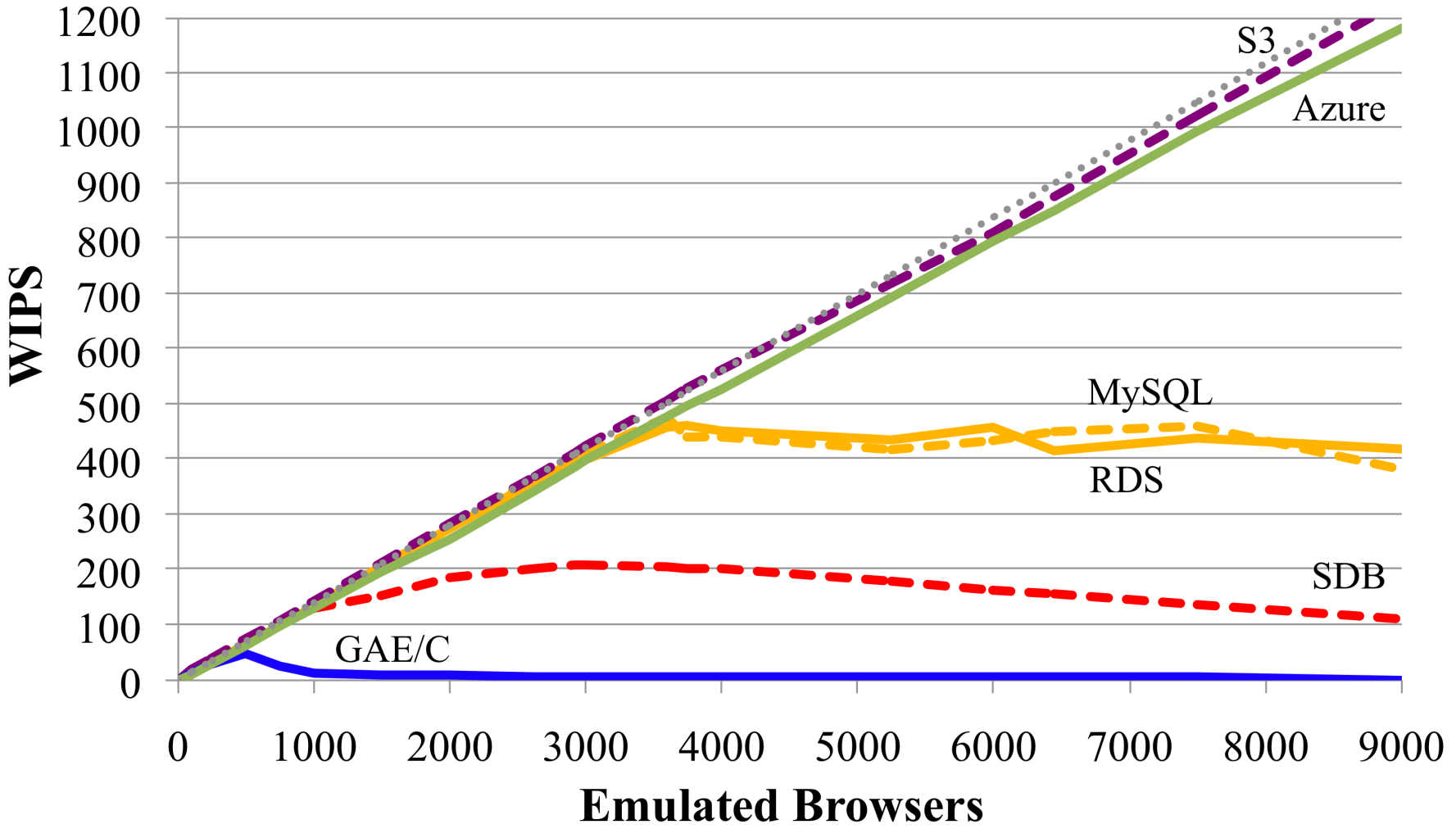
Throughput [WIPS] – Strongly Consistent Services



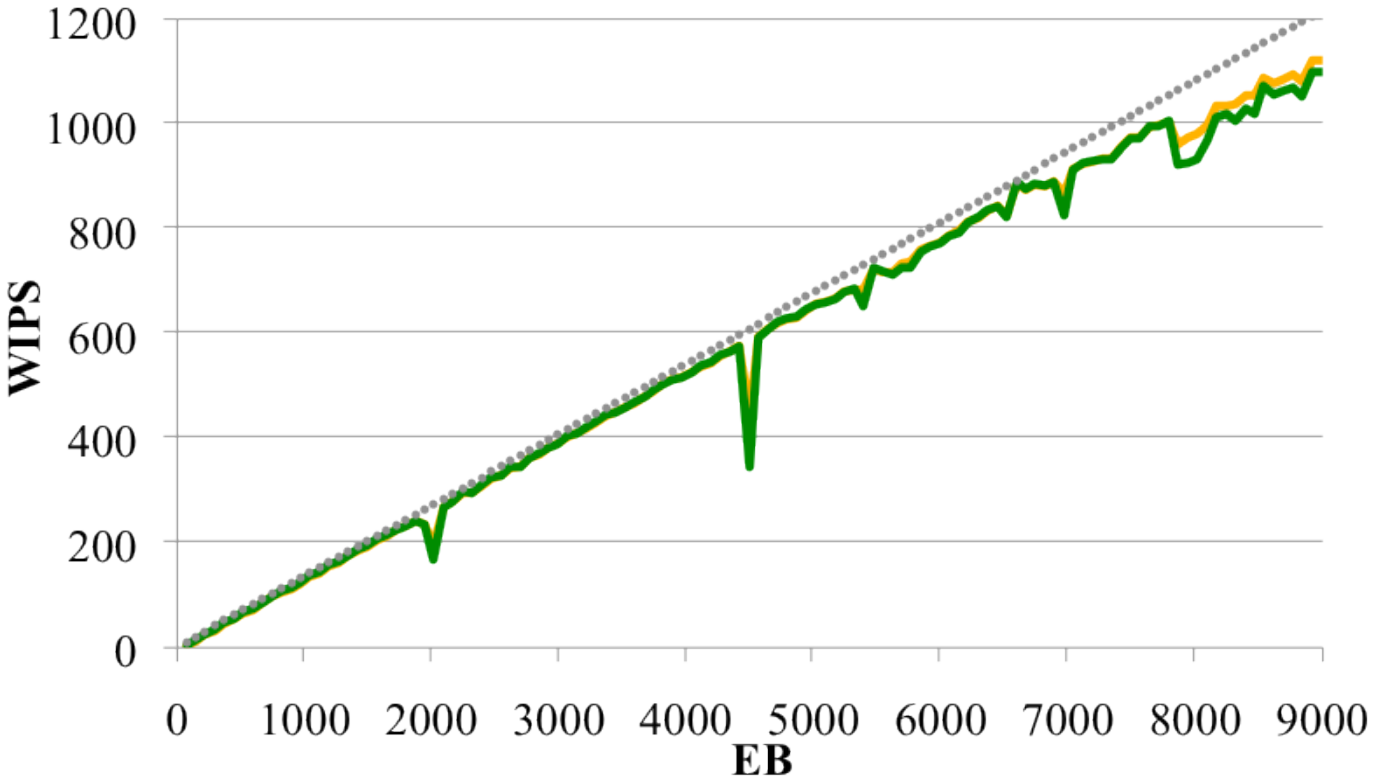
Throughput [WIPS] – Weakly Consistent Services



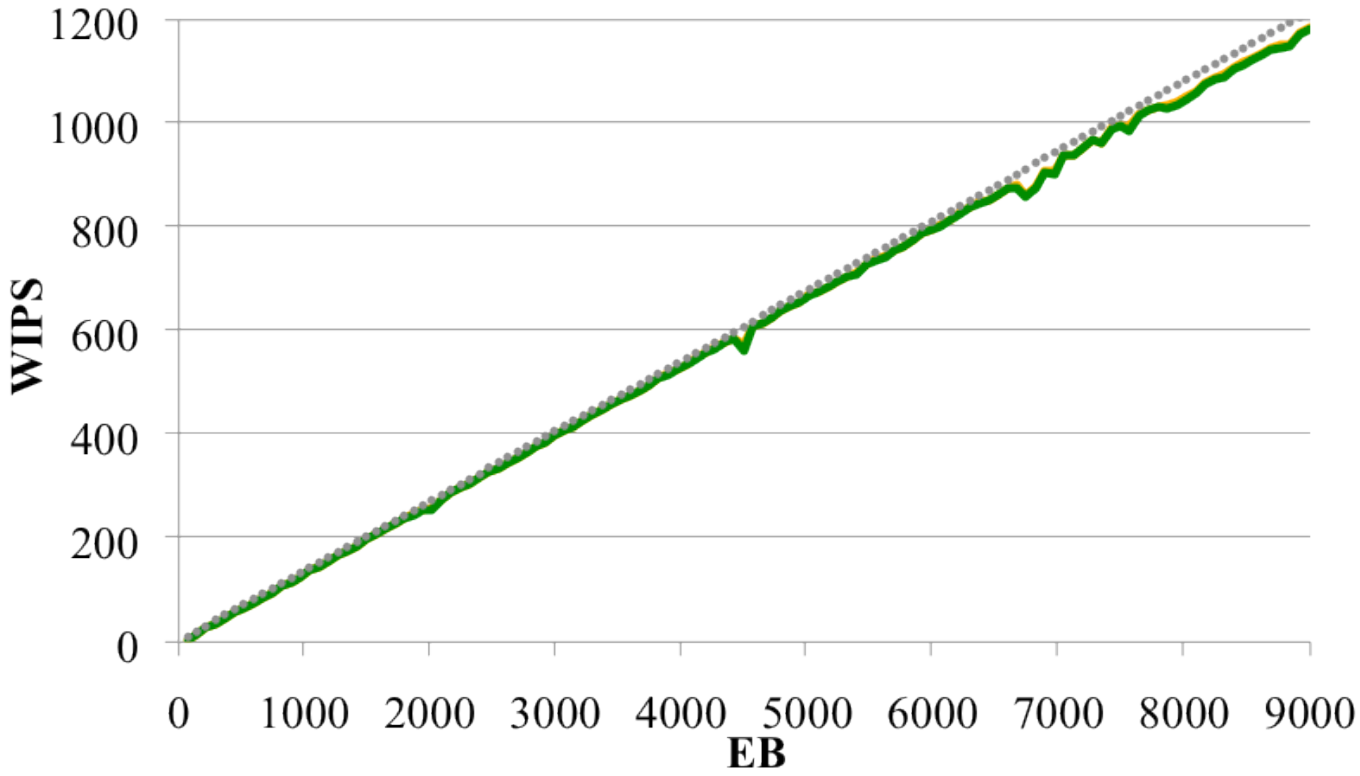
Throughput [WIPS]



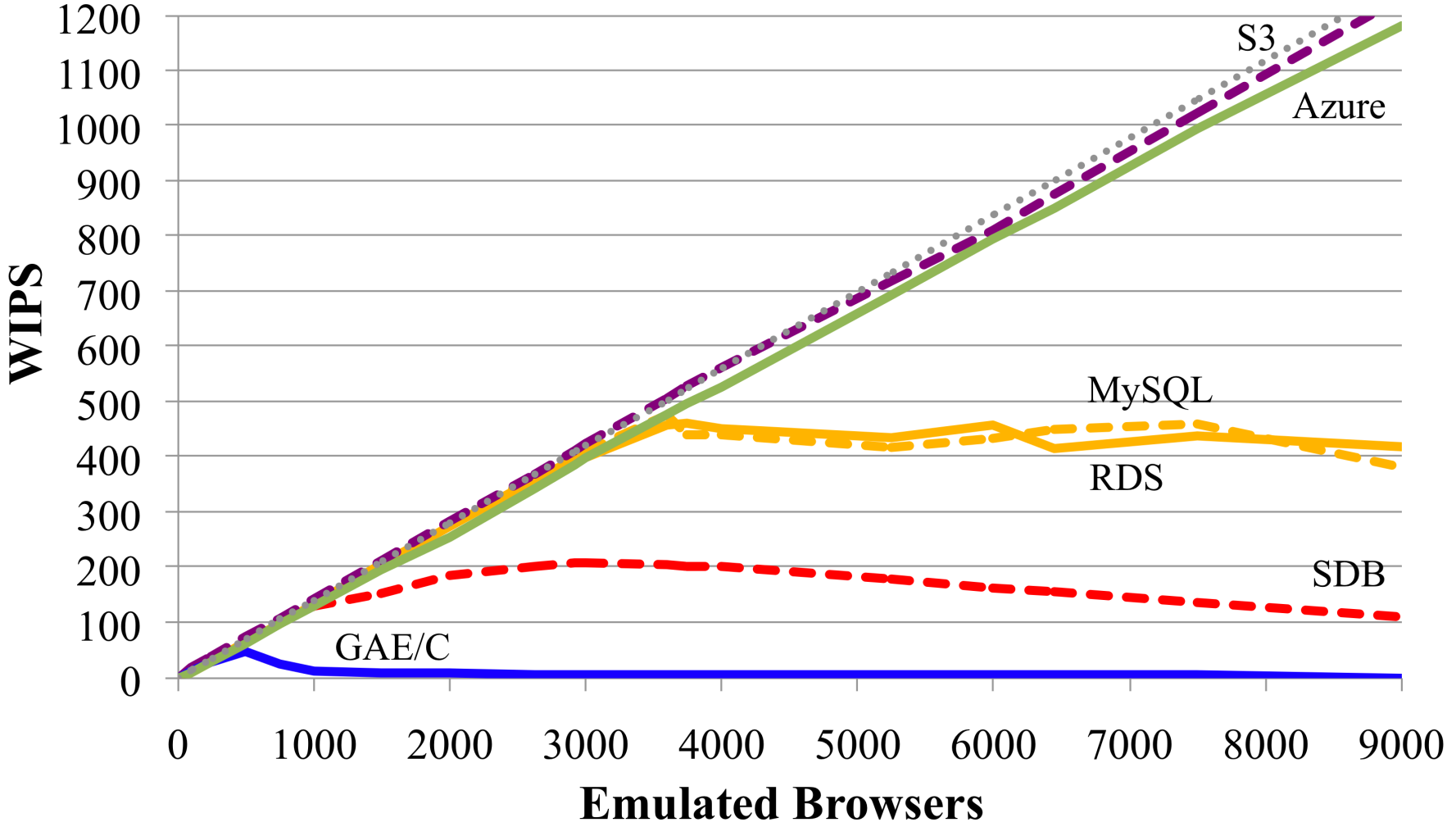
Scalability Azure (first few runs)



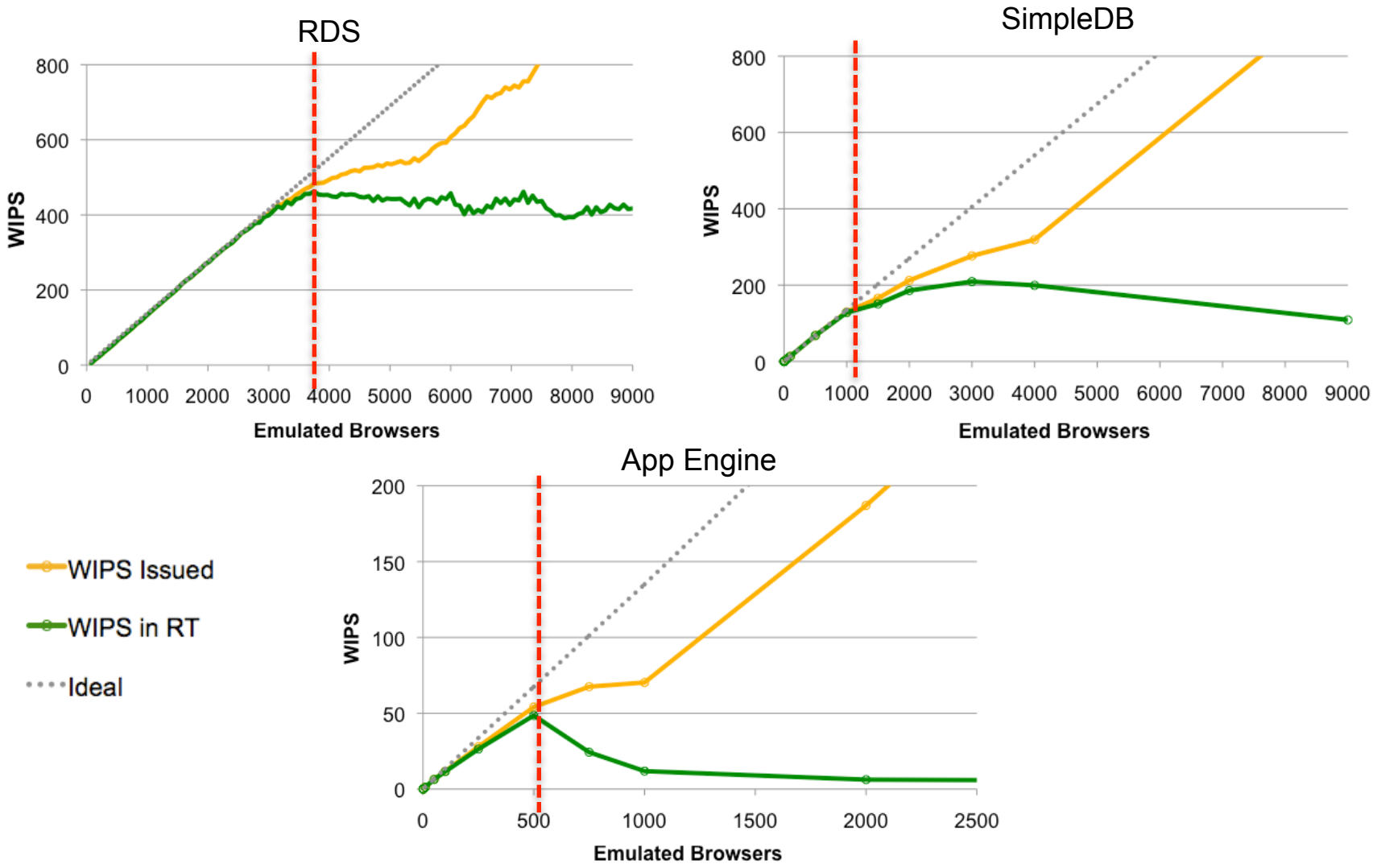
Scalability Azure (final results)



Throughput [WIPS]



Overload Behavior



Cost [m$/WI]

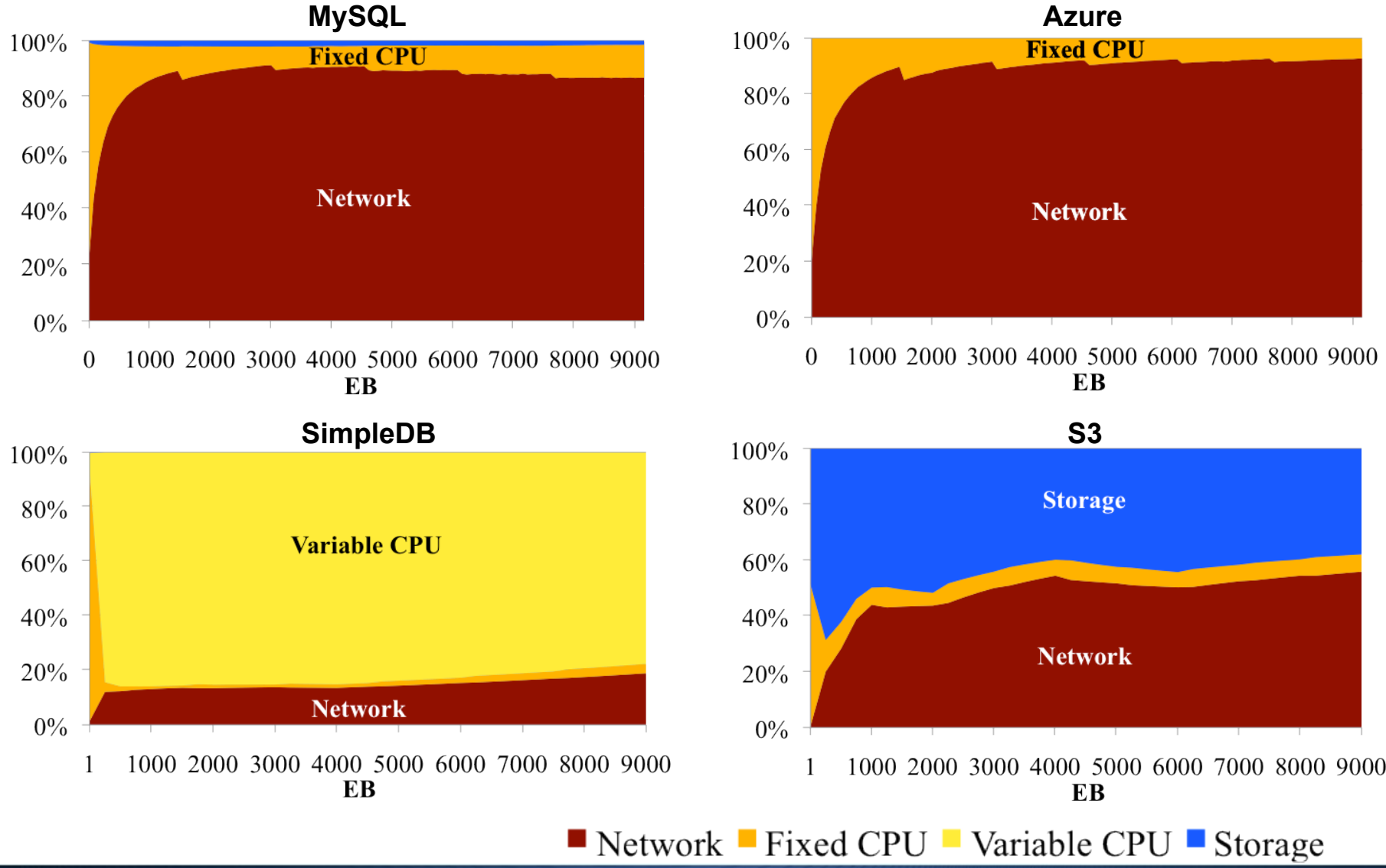
	EBs					Fully Utilized
	1	10	100	500	1000	
MySQL	0.635	0.072	0.020	0.006	0.006	0.005
MySQL/R	2.334	0.238	0.034	0.008	**0.006**	**0.005**
RDS	1.211	0.126	0.032	0.008	0.006	0.005
SimpleDB	0.384	0.073	0.042	0.039	0.037	0.037
S3	1.304	0.206	0.042	0.019	0.011	0.009
Google AE	0.002	0.028	0.033	0.042	0.176	0.042
Google AE/C	**0.002**	0.018	0.026	0.028	0.134	0.028
Azure	0.775	0.084	0.023	0.008	0.006	0.005

Cost predictability [m$/WI]

	mean ± *s*
MySQL	0.015 ± 0.077
MySQL/R	0.043 ± 0.284
RDS	0.030 ± 0.154
SimpleDB	0.063 ± 0.089
S3	0.018 ± 0.098
Google AE	0.029 ± 0.016
Google AE/C	0.021 **± 0.011**
Azure	0.010 ± 0.058

- In an optimal scenario the Cost / WI is constant and independent of the load
- *s:* the lower the better
- Google‘s price model fits best the pay-as-you-go paradigm!

Relative Cost Factors



Questions/Comments we got…

- You are using Google AppEngine not correctly! Google's TRX model is like getting a lock on the whole DB (i.e., entity group)
- Why do you not partition the data for GAE and SimpleDB? It would make them scale
- Why does Azure scale so well? Will it continue to scale?
- This is not a fair comparison (because of consistency, functionality, purposes of the architecture,…)!
- Why stop at 9000 Ebs?

Questions/Comments we got… (ctd)

- Which platform was the easiest to program?
- The data size is too small. Why not use a bigger data size?
- MS Azure is not better than SimpleDB if you increase the data size.
- TPC-W is not the right choice. For a different workload it would look different.
- Can you include product X in your study?
- Is the source code available?

Personal remarks on SQL Azure…

- Really easy-to-use platform
- SQL Azure gives the comfort of a traditional DB and the fault-tolerance of a cloud service
- However, strong limitations in the data size and scalability and the pricing model makes it best suited for medium-sized deployments (i.e., for small applications to expensive, for really large applications not scalable)
- What we would like to see: Automatic scaling of VMs, Java support, low-level access,…

Conclusion & Future Work

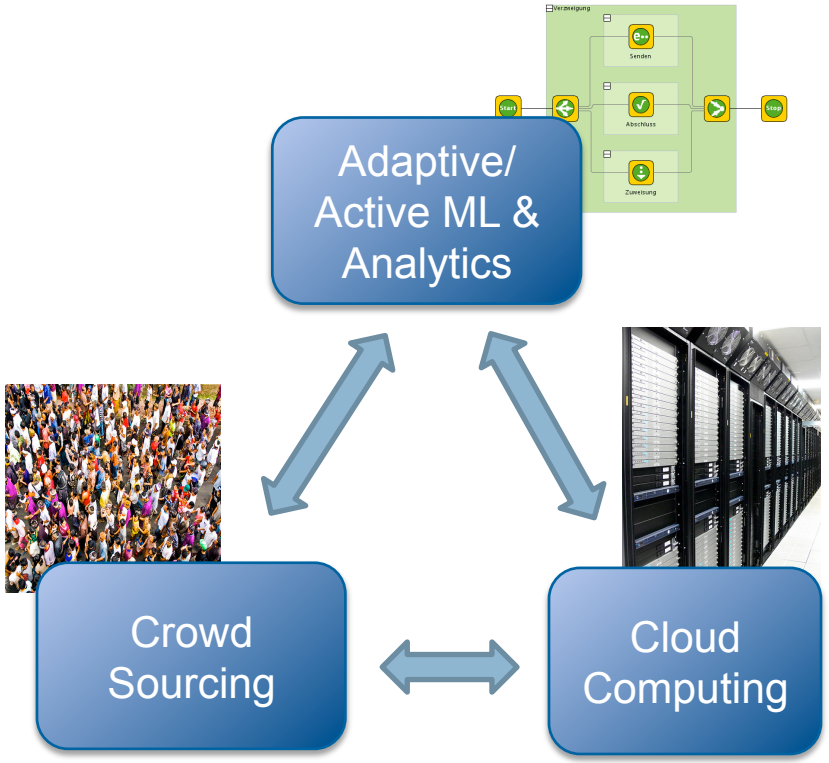
- Have we started a new benchmark war?
  - We have measured elasticity, cost and cost predictability
  - Vendors implement different architectures having significant effects on performance
  - **Discussion available on http://www.pubzone.org**

- Now we want to find the silver bullet for data management in the cloud
  - Develop a reference architecture
  - Combine techniques of partitioning + replication + distributed control
  - **http://www.systems.ethz.ch/research/projects/projectslist#Cloudy**

AMP Lab: The Next Generation

Algorithms, Machines and People

- **Observation**: We can extract only a small fraction of the value from the wealth of data available.
- **Mission**: Enable **many people** to collaborate to collect, generate, clean, make sense of and **utilize lots of data.**
- **Approach**: A **holistic view** of the stack from data visualization down to cluster & multi-core support.
- A five year plan; will dovetail with RADLab completion



For more information – catch me later
or send mail: franklin@cs.berkeley.edu / kraska@cs.erkely.edu