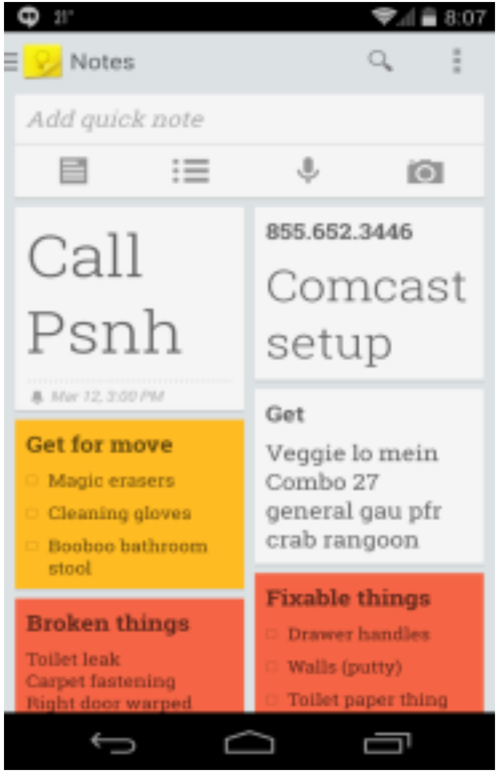
WebPerf: Evaluating “What-If” Scenarios for Cloud-hosted Web Applications

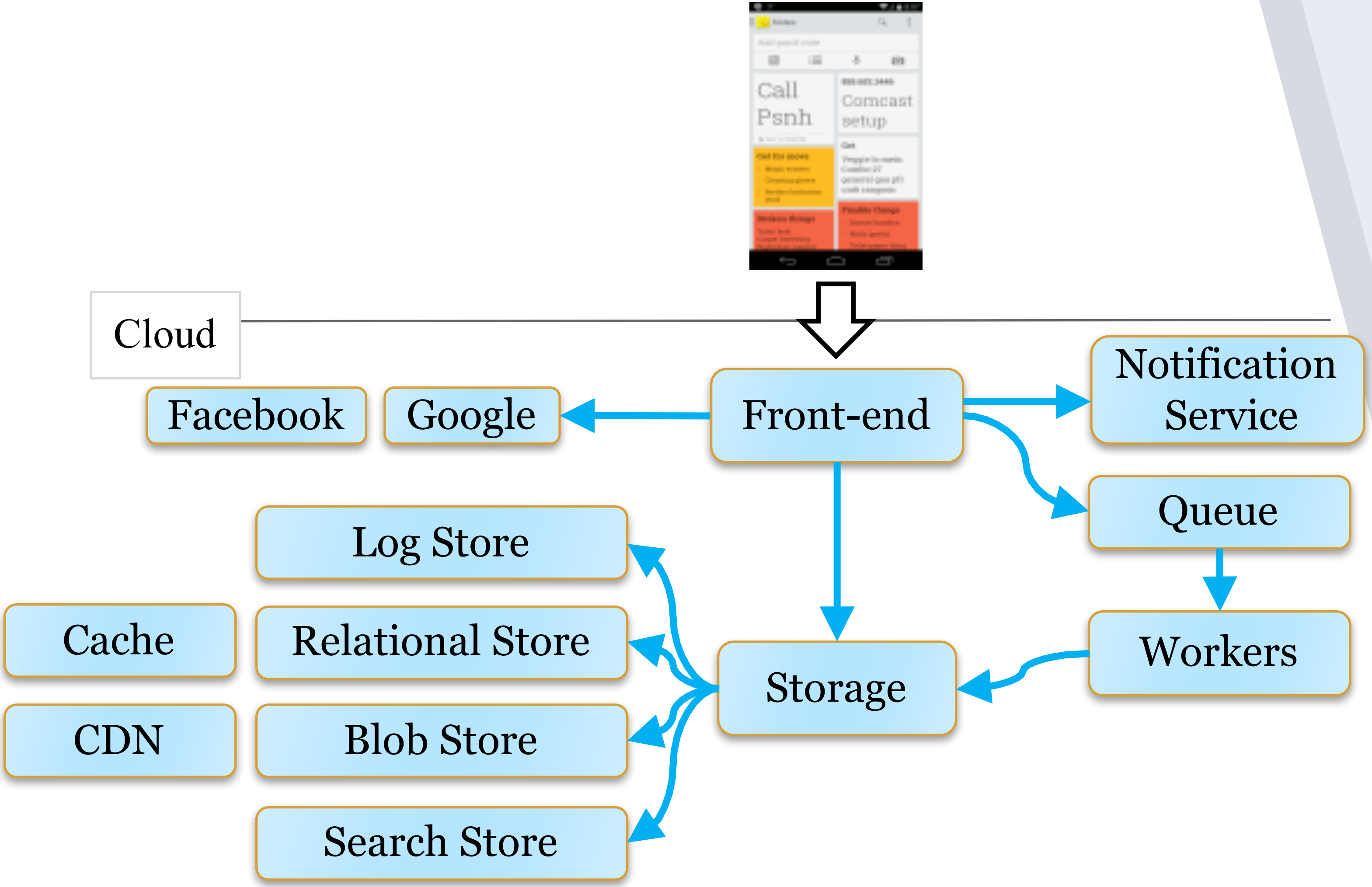
Yurong Jiang
Lenin Ravindranath
Suman Nath
Ramesh Govindan

Microsoft Research
USC University of Southern California

A Cloud-Hosted Web Application



Modern Cloud Applications are Complex



Cloud-side Latency

~~Latency~~ of these applications is critical for user experience

Developers find it hard to optimize cloud-side latency for cloud-hosted Web applications

Front-end

INSTANCE	CORES	RAM	DISK SIZES	PRICE
A0	1	0.75 GB	19 GB	\$0.02/hr (~\$15/mo)
A1	1	1.75 GB	224 GB	\$0.08/hr (~\$60/mo)
A2	2	3.5 GB	489 GB	\$0.16/hr (~\$119/mo)
A3	4	7 GB	999 GB	\$0.32/hr (~\$238/mo)
A4	8	14 GB	2,039 GB	\$0.64/hr (~\$476/mo)

Each choice impacts latency

Configuration Complexity

Relational Store

Azure SQL

Premium

	DTUs[2]	MAX STORAGE PER DB	PRICE[1]
P1	125	500 GB	~$465/mo
P2	250	500 GB	~$930/mo
P4	500	500 GB	~$1,860/mo
P6	1000	500 GB	~$3,720/mo
P11	1750	1 TB	~$7,001/mo

Latency implications hard to understand

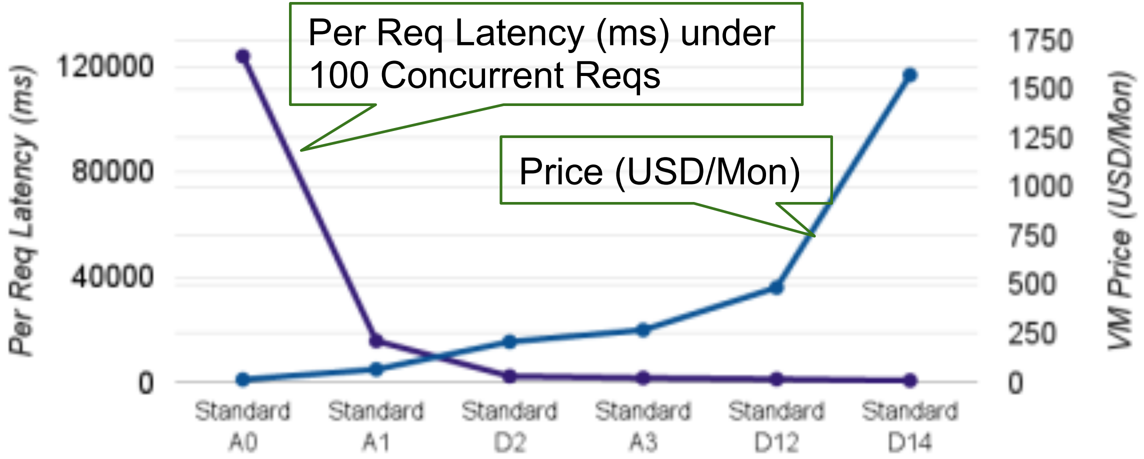
What if?

What if I move the blob store from basic to standard tier?

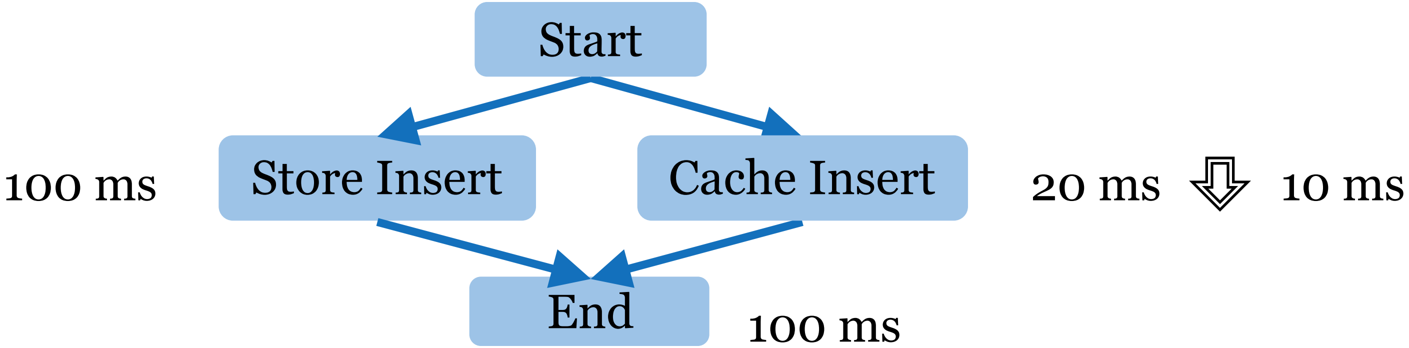
Cloud App + Service Configuration = 100 ms

$30 ⇒ $100

Answer to what-if question may depend on workload



Answer to what-if question may depend on causal dependencies



A what-if capability should be expressive

Relational Store 

What if I re-locate this component?

Relational Store 

What if I increase this component's load?

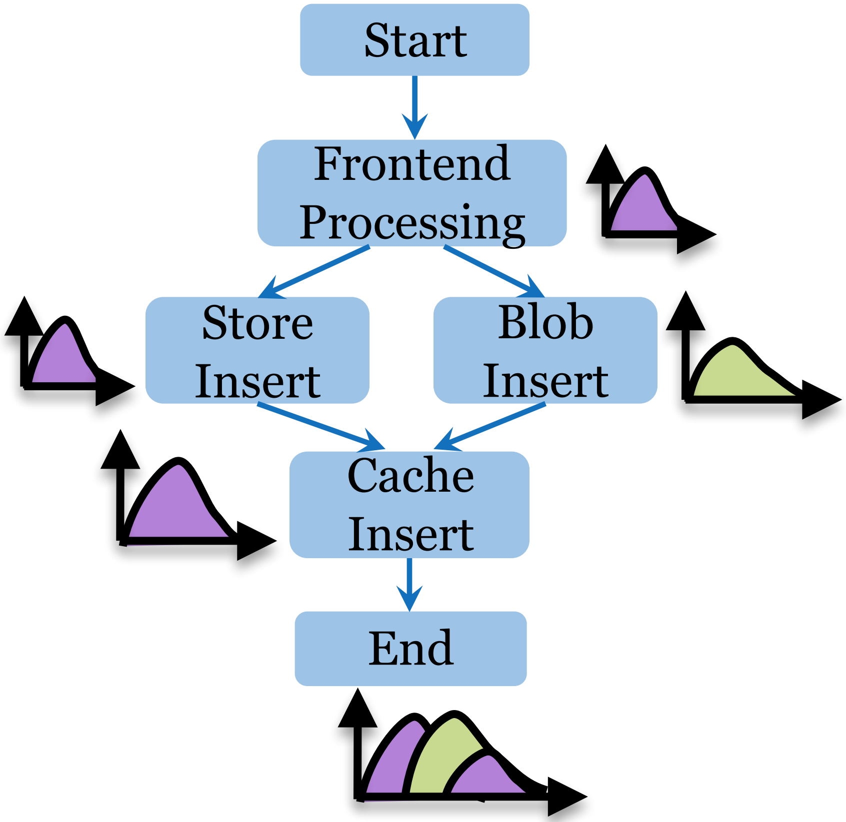
Table Store 

What if a replica fails?

WebPerf is a what-if scenario evaluator

- ❖ Input: a what-if scenario
- ❖ Output: resulting **cloud-side latency distribution**

What if I upgrade blob storage from basic to standard tier?



Dependency graph extraction

Baseline latency estimation

Component Profiling

Cloud-side latency estimation

Developer supplies workload

Computed Offline

Why might WebPerf work?

Cloud deployments well-engineered

- Components designed for predictable latency
- Often co-located in same datacenter

- Cheap
- Fast
- Low Effort

Automate most steps

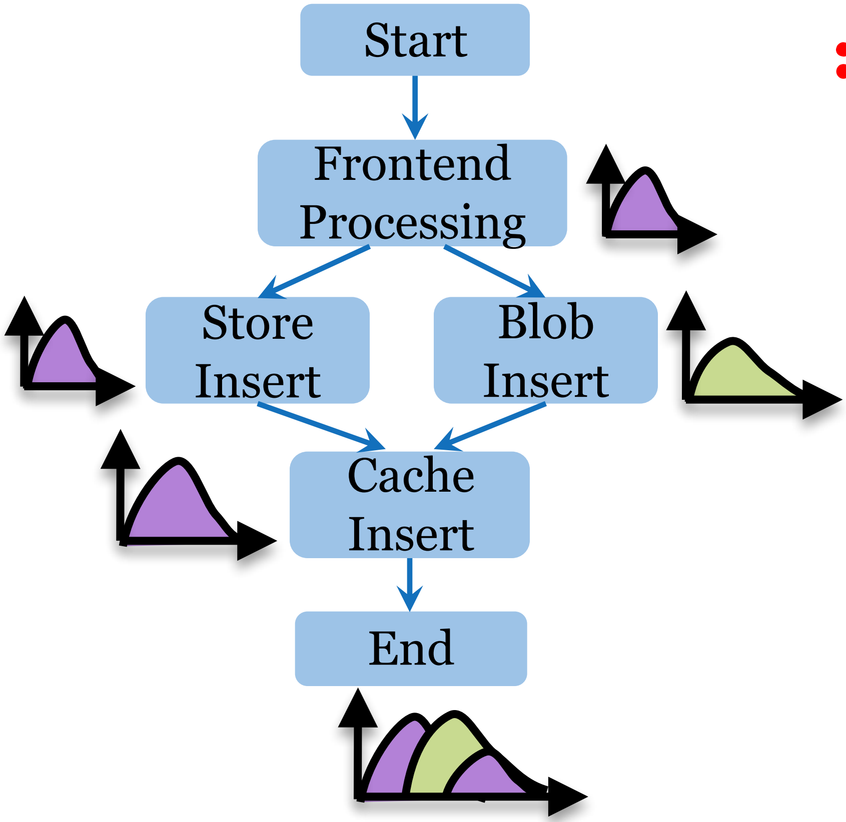
Why?

Many component profiles are application-independent

Compute offline, reuse

The dependency graph is usually independent of what-if scenario

Compute once



What if I upgrade blob storage from basic to standard tier?

- Dependency graph extraction
- Baseline latency estimation
- Component Profiling
- Cloud-side latency estimation

Goal

Fast, accurate dependency extraction with *zero developer input*

Approach

Track dependencies at run-time by *instrumenting binary*

Prior work has not considered this

Task Asynchronous Programming

- ❖ Many cloud apps use this
- ❖ Only mechanism for asynchronous I/O in Azure
- ❖ AWS provides APIs for .NET

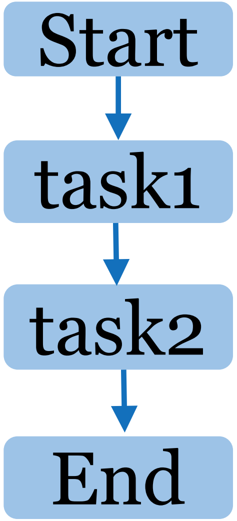
```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```

On front-end

Start task

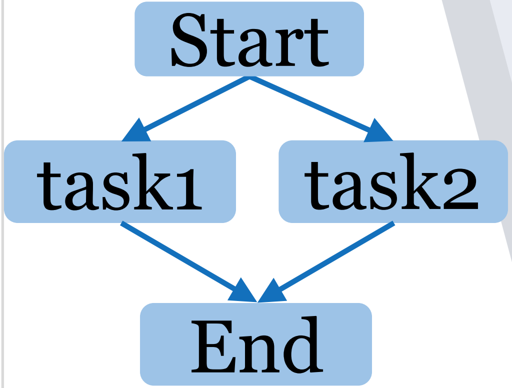
Continue

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```



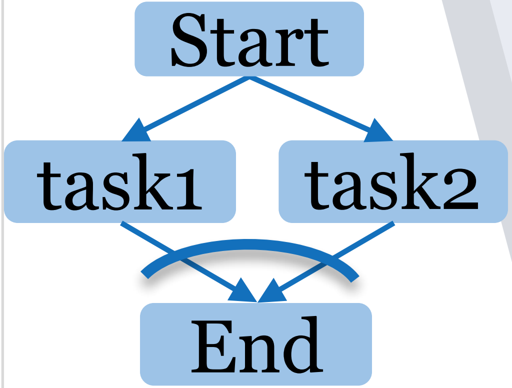
WhenAll: Continue only when all tasks finish

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    task2 = cache.get(key2);
    value1, value2 = await
            Task.WhenAll(task1, task2);
    /* construct response */
    return response;
}
```



WhenAny: Continue when *any one* task finishes

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    task2 = cache.get(key2);
    value1, value2 = await
            Task.WhenAny(task1, task2);
    /* construct response */
    return response;
}
```



```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```

Init

Received value1

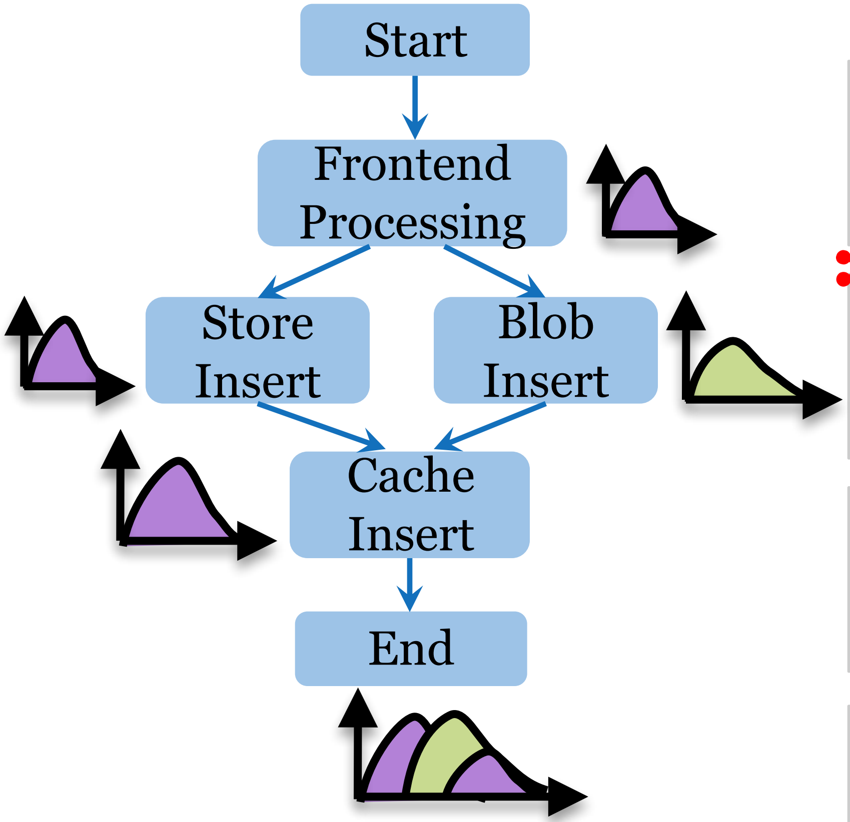
Received value2

.NET compiler generates this

Instrument state machine *binary* to *dynamically* track tasks and continuations

WebPerf Approach

What if I upgrade blob storage from basic to standard tier?



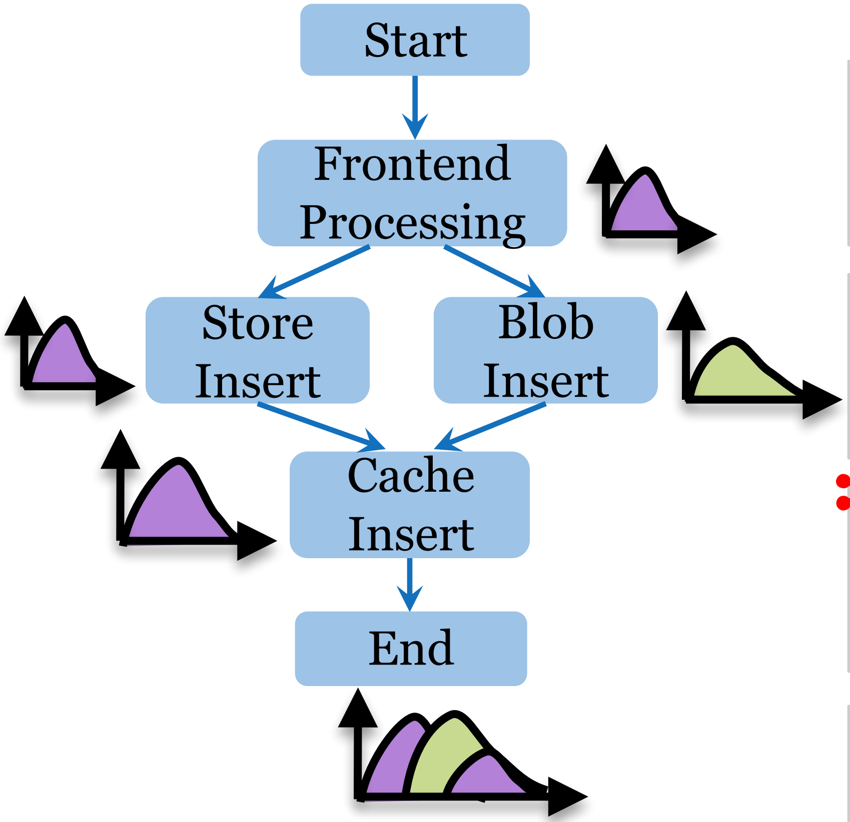
Dependency graph extraction

Baseline latency estimation

Component Profiling

Cloud-side latency estimation

What if I upgrade blob storage from basic to standard tier?



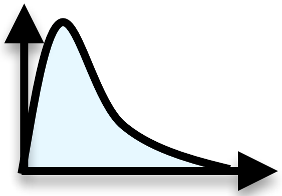
Dependency graph extraction

Baseline latency estimation

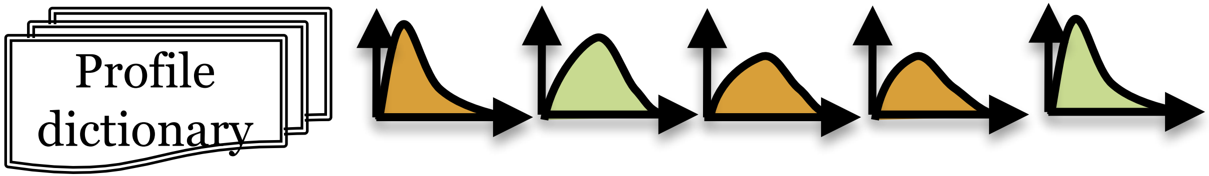
Component Profiling

Cloud-side latency estimation

A component's *profile* contains latency distributions of API calls to component

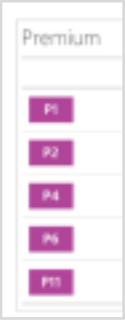


WebPerf profiles commonly used components *offline*



Generalized Profiles

Relational Store



Tiers

Relational Store

Location

Relational Store

Load

Table Store

Failure

Not all profiles can be computed offline

Relational Store

Azure SQL

SQL join latency depends on size

Cache

Redis Cache

Cache latency depends on hit rate

Relational Store

Sharded TableStore

Access latency depends on skew

WebPerf uses *parameterized profiles*

- User must specify *workload hint*

Relational Store
Azure SQL

Size

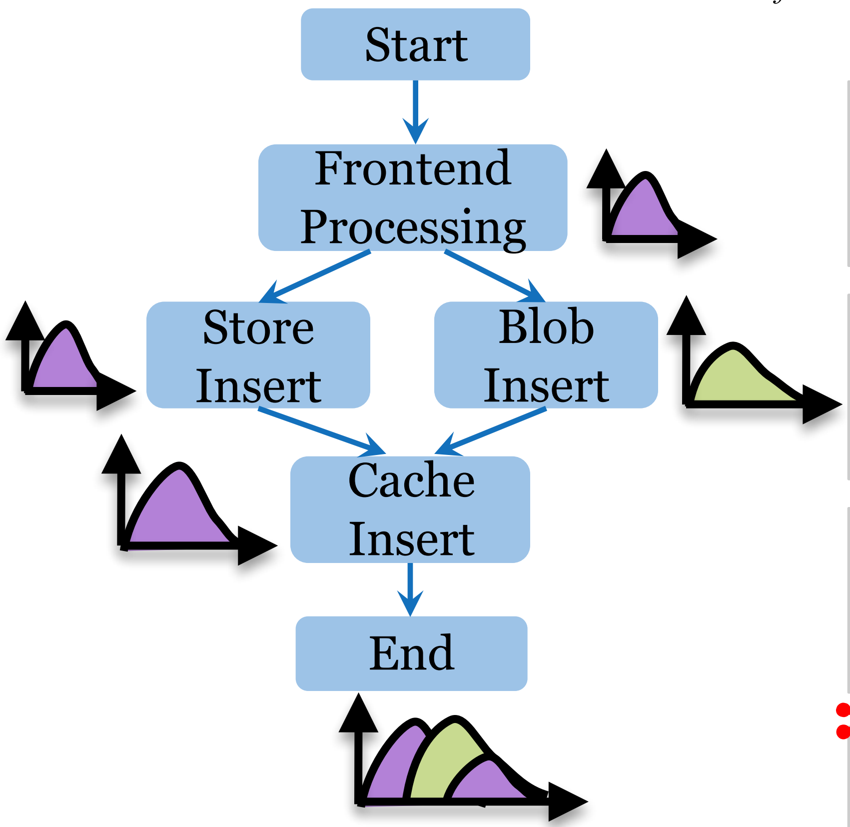
Cache
Redis Cache

Hit rate

Relational Store
Sharded TableStore

Skew

What if I upgrade blob storage from basic to standard tier?



Dependency graph extraction

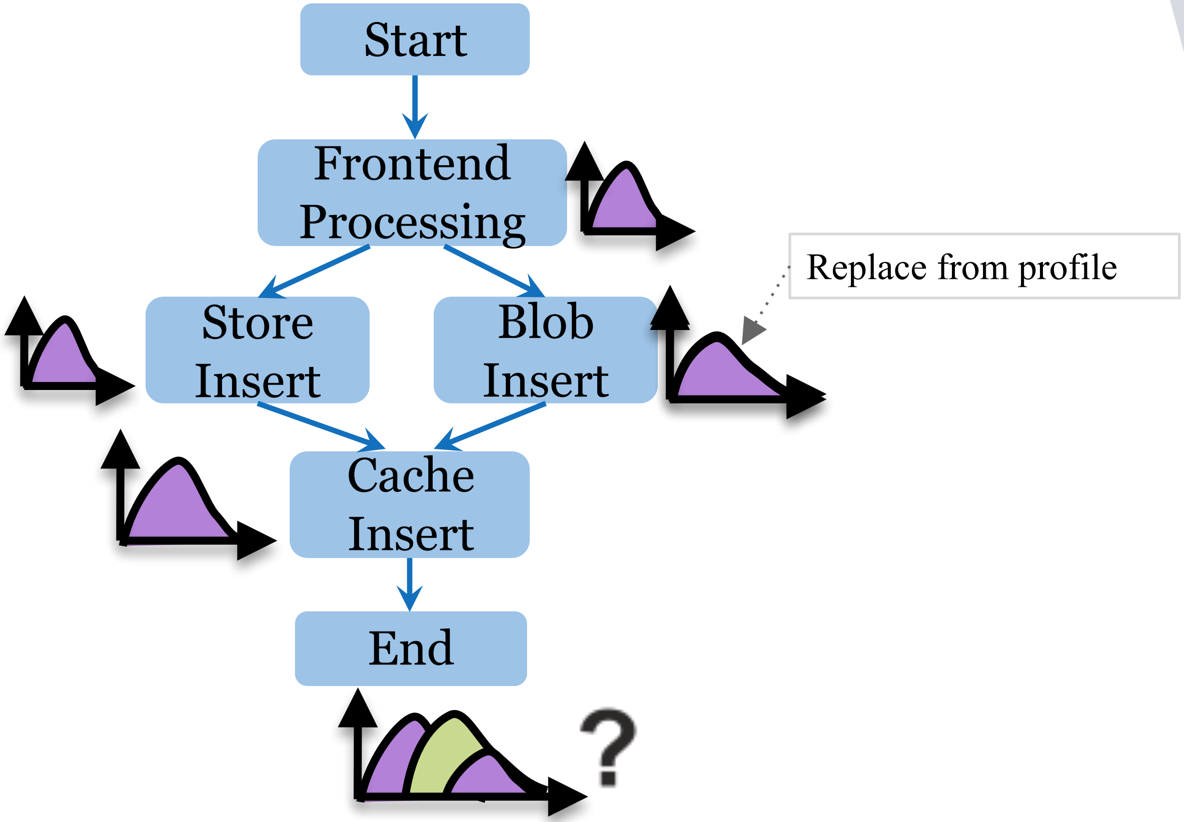
Baseline latency estimation

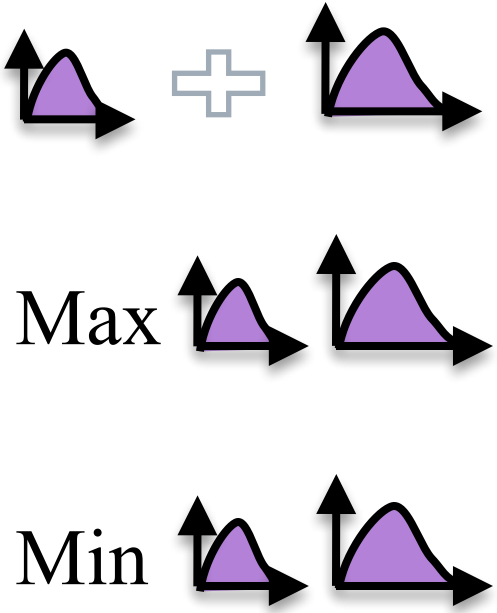
Component Profiling

Cloud-side latency estimation

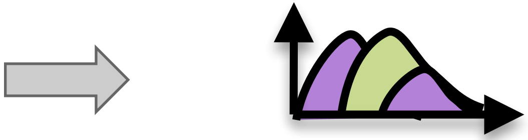
Cloud-Side Latency Estimation

What if I upgrade blob storage from basic to standard tier?





Simple operations on distributions suffice



WebPerf is *accurate, fast, cheap,* and *requires low developer effort*

- ❖ How accurate is WebPerf?
- ❖ Are workload hints necessary?

Different functionality

Different components

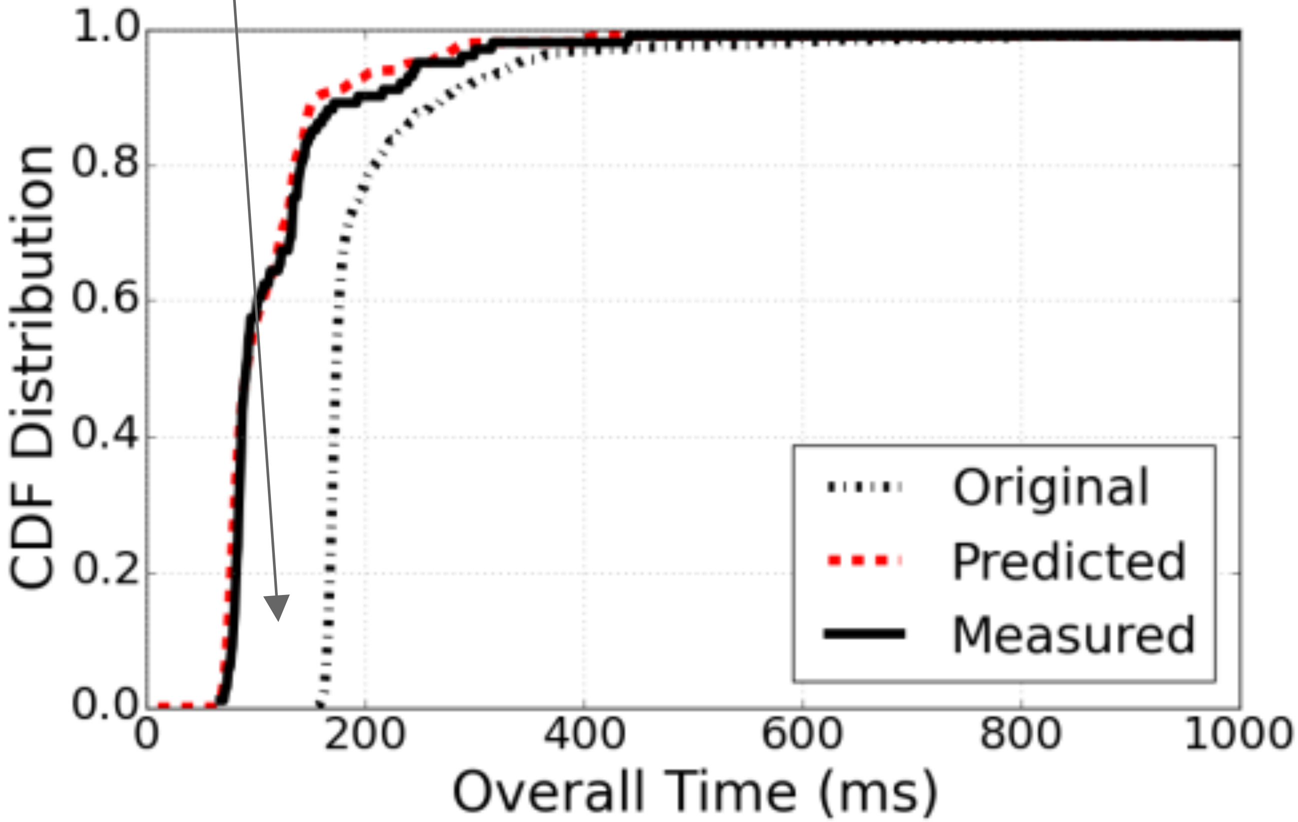
Varying complexity

Application	Azure components used	Average I/O Calls
SocialForum	Blob storage, Redis cache, Service bus, Search, Table	116
SmartStore.Net	SQL	41
ContosoAds	Blob storage, Queue, SQL, Search	56
EmailSubscriber	Blob storage, Queue, Table	26
ContactManager	Blob storage, SQL	8
CourseManager	Blob storage, SQL	44

What-if scenario	Example
Tier: A component X is upgraded to tier Y	X = A Redis cache, Y = a standard tier (from a basic tier)
Load: X concurrent requests to component Y	X = 100 , Y = the application or a SQL database
Interference: CPU and/or memory pressure, from collocated applications, of X%	X = 50% CPU, 80% memory
Location: A component X is deployed at location Y	X = A Redis Cache or a front end, Y = Singapore
Failure: An instance of a replicated component X fails	X = A replicated front-end or SQL database

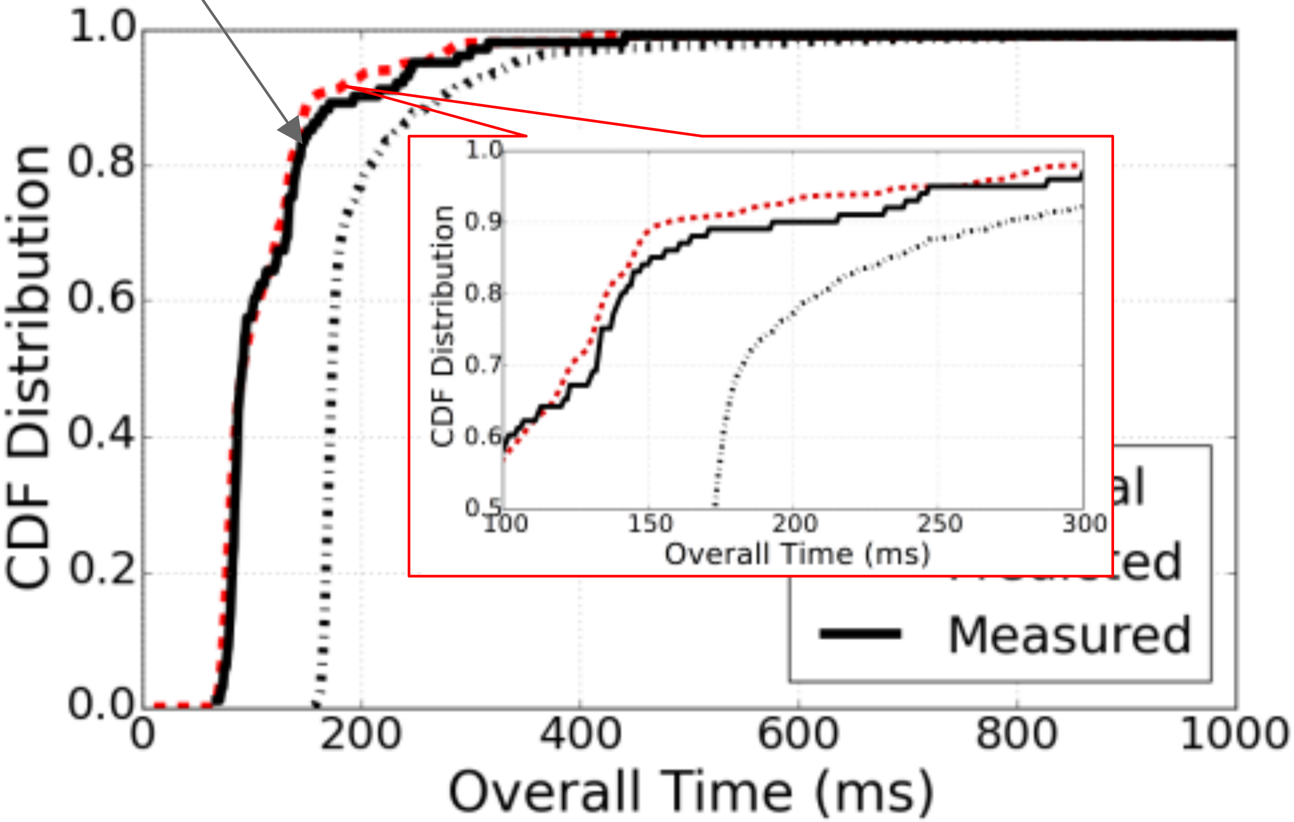
Configuration choices can significantly impact latency

What if I move the Redis cache in SocialForum from basic to standard tier?



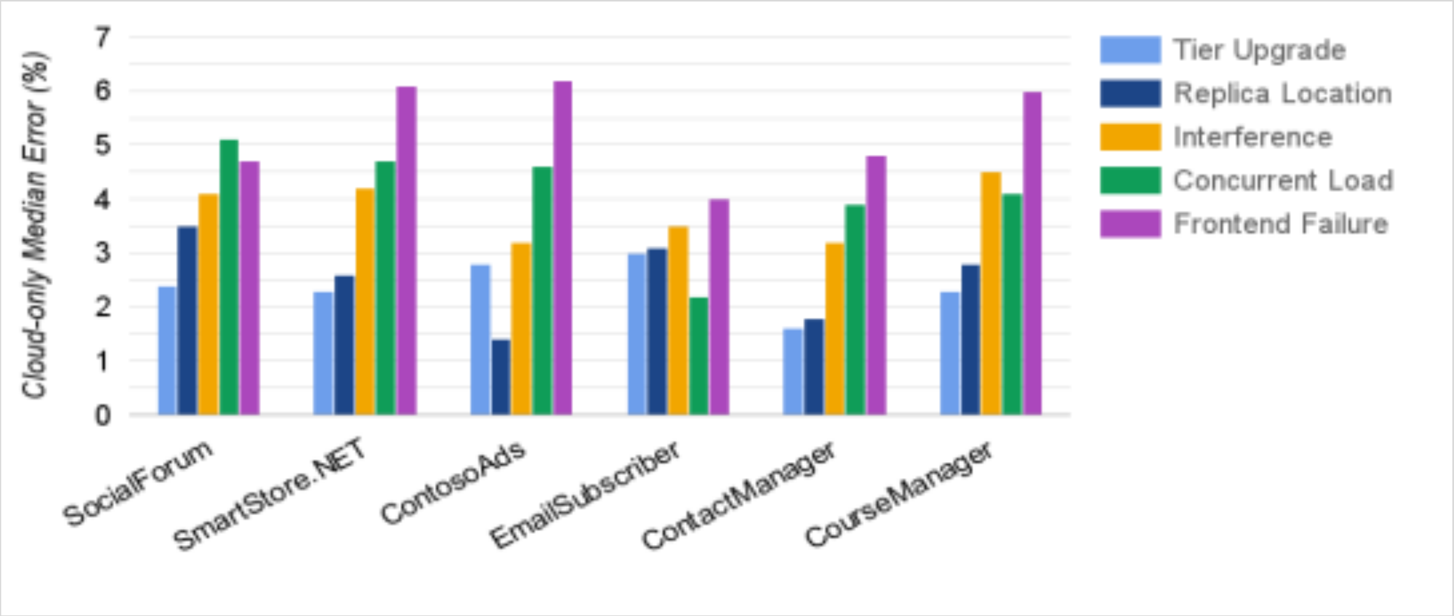
Prediction closely matches ground truth

What if I move the Redis cache from basic to standard tier?



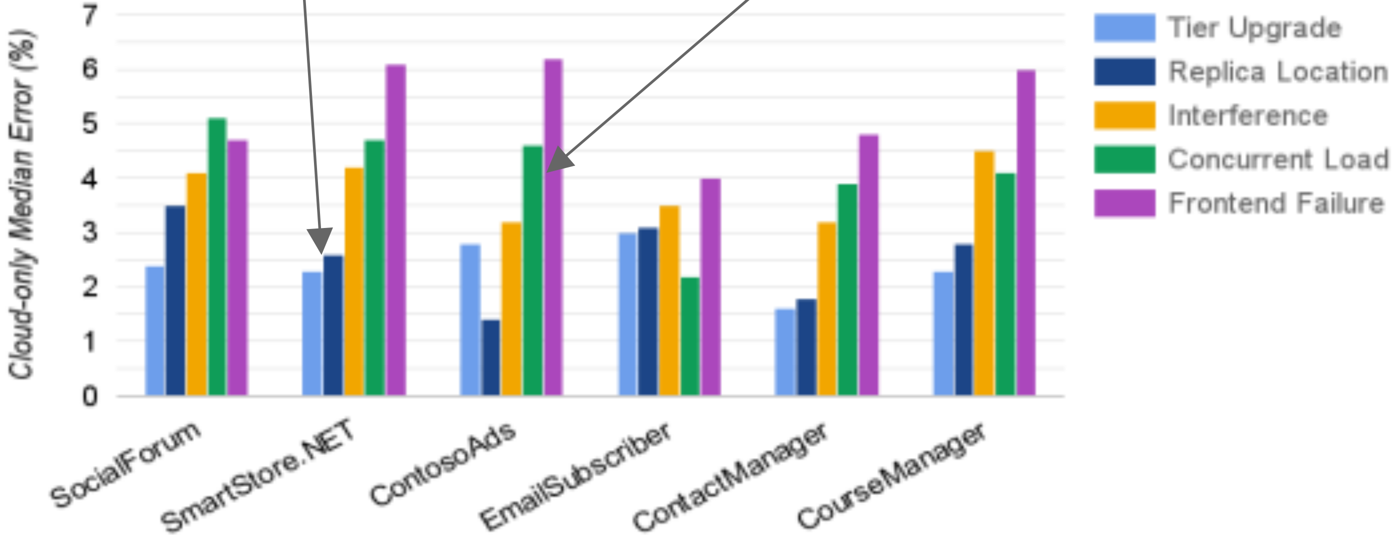
Median prediction error under 7%

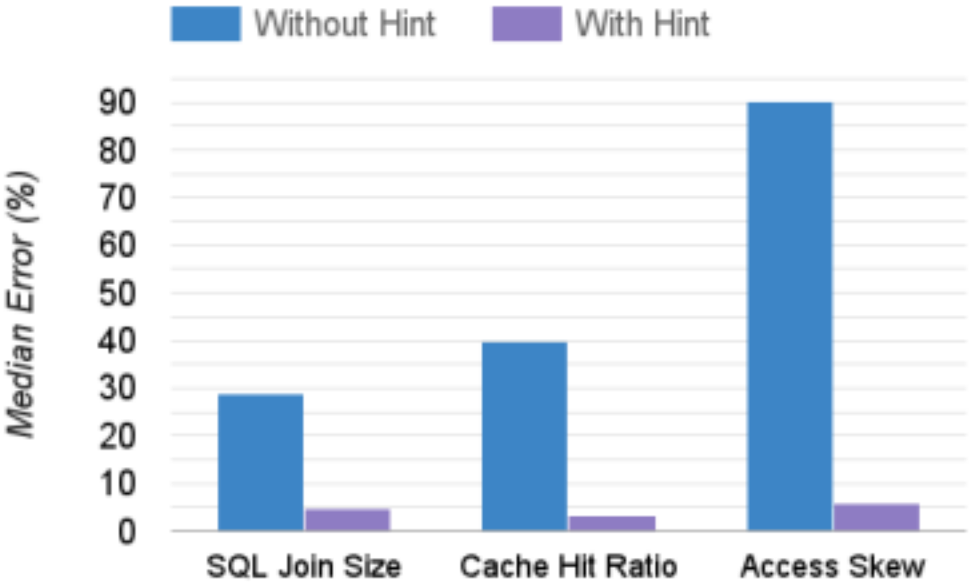
Difference between predicted distribution and ground truth



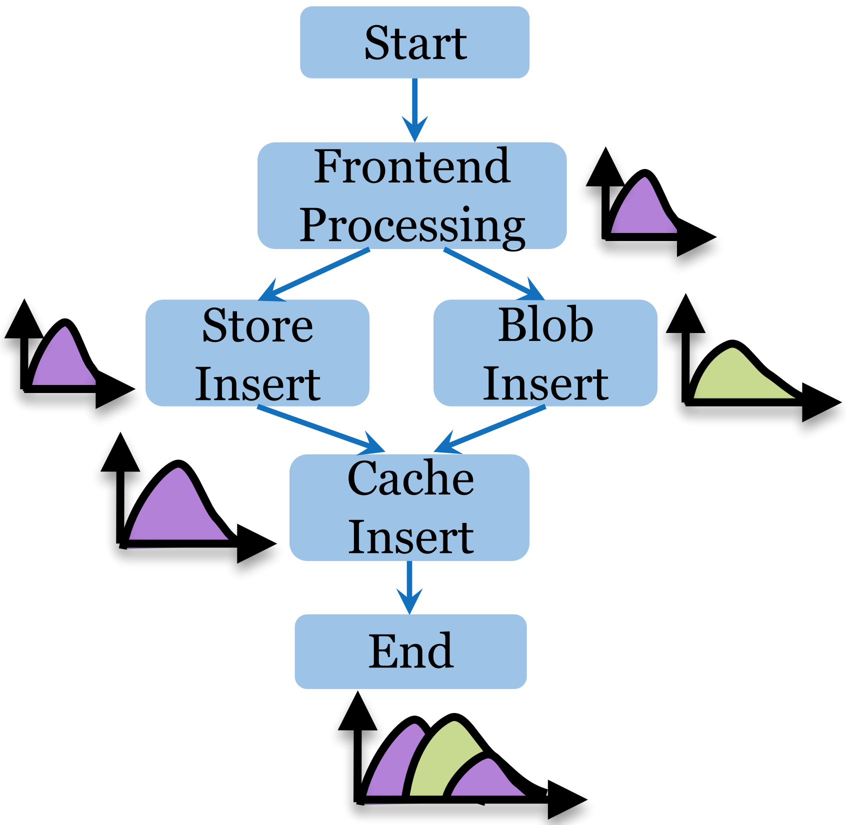
Low median error for tier and replication

Slightly higher error for load and failure





Workload hints can significantly improve accuracy



WebPerf predicts cloud-side latency distributions for different what-if scenarios

It accurately tracks dependencies and profiles components offline

Across six different applications and scenarios, its error is less than 7%

WebPerf Contributions and Summary

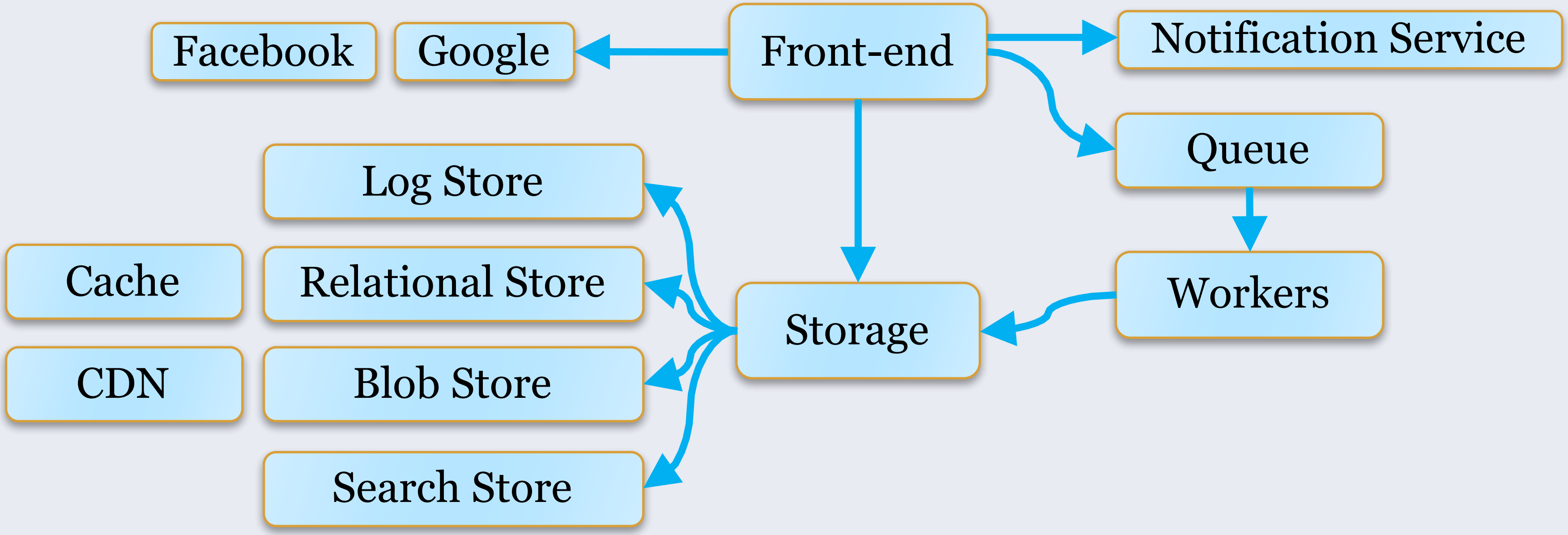
An automated tool to instrument web apps and capture both browser objects and front end cloud processing dependency

Predicting web app cloud latency and end-to-end latency in probabilistic setting under six different scenarios

Evaluations with six real websites show WebPerf achieves < 7% median prediction error

Thank you

Large number of configuration choices

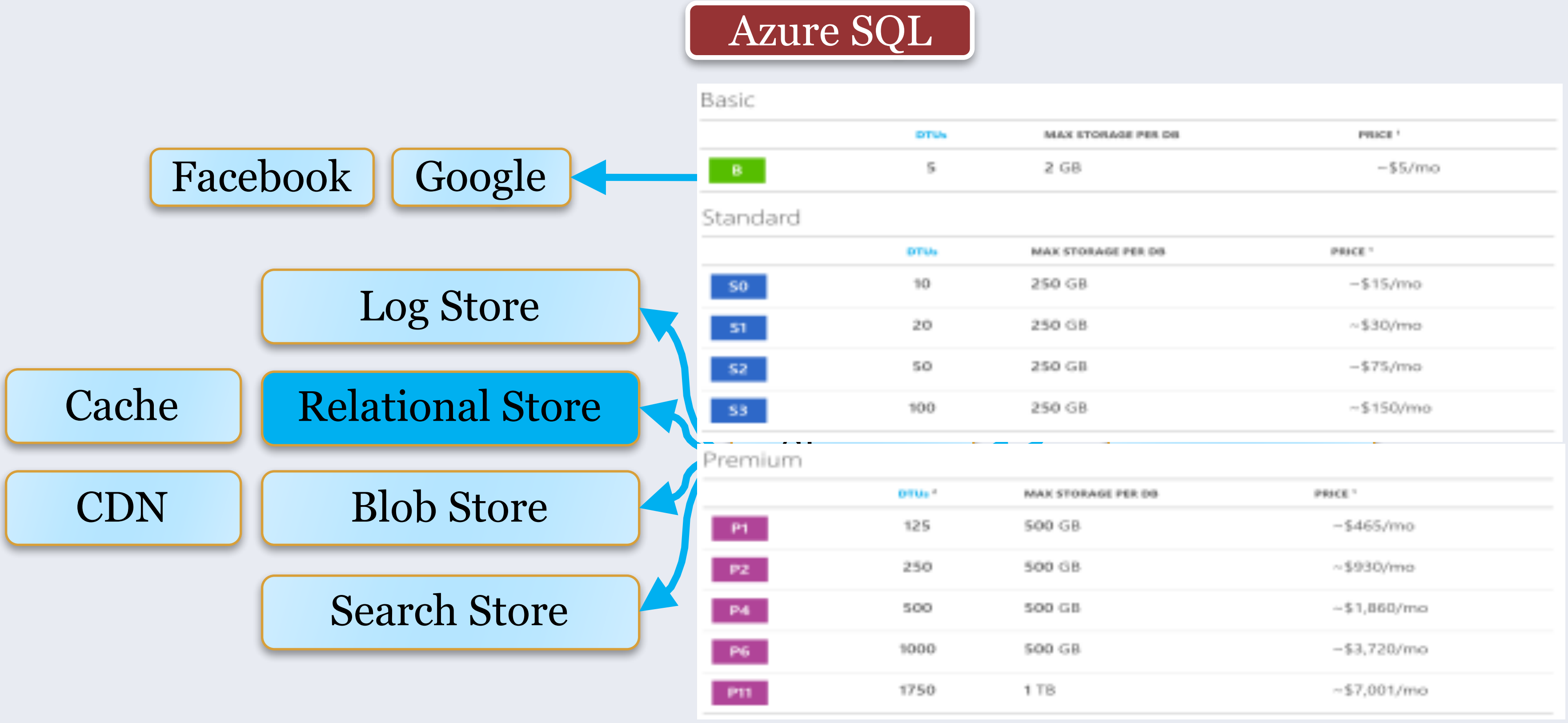


Large number of configuration choices

INSTANCE	CORES	RAM	DISK SIZES	PRICE
A0	1	0.75 GB	19 GB	$0.02/hr (~$15/mo)
A1	1	1.75 GB	224 GB	$0.08/hr (~$60/mo)
A2	2	3.5 GB	489 GB	$0.16/hr (~$119/mo)
A3	4	7 GB	999 GB	$0.32/hr (~$238/mo)
A4	8	14 GB	2,039 GB	$0.64/hr (~$476/mo)
A5	2	14 GB	489 GB	$0.35/hr (~$260/mo)
A6	4	28 GB	999 GB	$0.71/hr (~$528/mo)
A7	8	56 GB	2,039 GB	$1.41/hr (~$1,049/mo)

D1	1	3.5 GB	50 GB	$0.14/hr (~$104/mo)
D2	2	7 GB	100 GB	$0.28/hr (~$208/mo)
D3	4	14 GB	200 GB	$0.56/hr (~$417/mo)
D4	8	28 GB	400 GB	$1.12/hr (~$833/mo)
D11	2	14 GB	100 GB	$0.33/hr (~$246/mo)
D12	4	28 GB	200 GB	$0.652/hr (~$485/mo)
D13	8	56 GB	400 GB	$1.173/hr (~$873/mo)
D14	16	112 GB	800 GB	$2.111/hr (~$1,571/mo)

Large number of configuration choices



Basic

	DTUs	MAX STORAGE PER DB	PRICE ¹
B	5	2 GB	~$5/mo

Standard

	DTUs	MAX STORAGE PER DB	PRICE ¹
S0	10	250 GB	~$15/mo
S1	20	250 GB	~$30/mo
S2	50	250 GB	~$75/mo
S3	100	250 GB	~$150/mo

Premium

	DTUs ²	MAX STORAGE PER DB	PRICE ¹
P1	125	500 GB	~$465/mo
P2	250	500 GB	~$930/mo
P4	500	500 GB	~$1,860/mo
P6	1000	500 GB	~$3,720/mo
P11	1750	1 TB	~$7,001/mo

Large number of configuration choices

Facebook

Google

Log Store

Cache

Relational Store

CDN

Blob Store

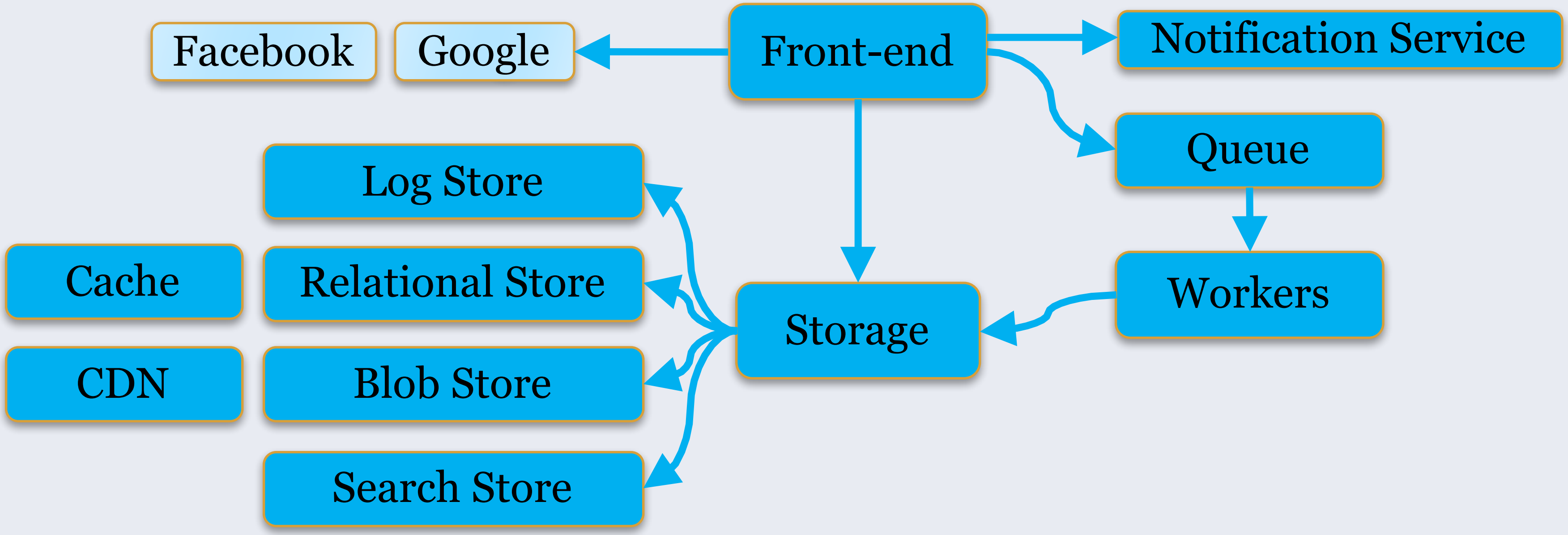
Search Store

Redis Cache

CACHE NAME	CACHE SIZE	BASIC	NETWORK PERFORMANCE	
C0	250 MB	$0.022/hr (~$16/mo)	Low	
C1	1 GB	$0.055/hr (~$41/mo)	Low	1000
C2	2.5 GB	$0.09/hr (~$67/mo)	Moderate	2000
C3	6 GB	$0.18/hr (~$134/mo)	Moderate	5000
C4	13 GB	$0.21/hr (~$156/mo)	Moderate	10000
C5	26 GB	$0.42/hr (~$312/mo)	High	15000
C6	53 GB	$0.84/hr (~$625/mo)	Highest	20000

CACHE NAME	CACHE SIZE	STANDARD	NETWORK PERFORMANCE	NUMBER OF CLIENT CONNECTIONS
C0	250 MB	$0.055/hr (~$41/mo)	Low	256
C1	1 GB	$0.138/hr (~$103/mo)	Low	1000
C2	2.5 GB	$0.225/hr (~$167/mo)	Moderate	2000
C3	6 GB	$0.45/hr (~$335/mo)	Moderate	5000
C4	13 GB	$0.525/hr (~$391/mo)	Moderate	10000
C5	26 GB	$1.05/hr (~$781/mo)	High	15000
C6	53 GB	$2.10/hr (~$1,562/mo)	Highest	20000
P2	13 GB	$1.11/hr (~$826/mo)	Moderate	15000
P3	26 GB	$2.219/hr (~$1,651/mo)	High	30000

Large number of configuration choices



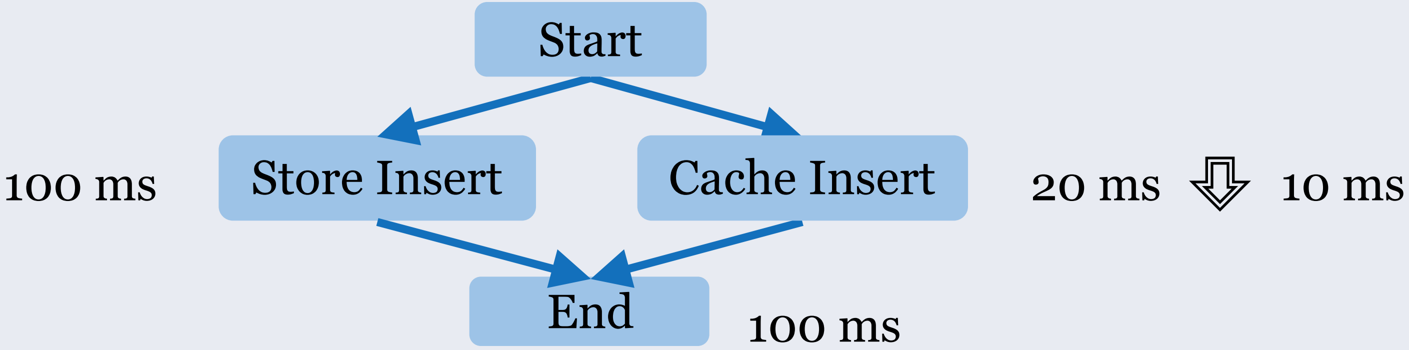
Reasoning about cost-performance trade-off is hard!

Cost-Performance Trade-off

- Configuration does not directly map to performance

INSTANCE	CORES	RAM	DISK SIZES	PRICE
A0	1	0.75 GB	19 GB	$0.02/hr (~$15/mo)
A1	1	1.75 GB	224 GB	$0.08/hr (~$60/mo)
A2	2	3.5 GB	489 GB	$0.16/hr (~$119/mo)
A3	4	7 GB	999 GB	$0.32/hr (~$238/mo)
A4	8	14 GB	2,039 GB	$0.64/hr (~$476/mo)

- End-to-end latency depends on application's causal dependency



“What-If” Analysis

What if?

Service Configuration

What if I move the front-end from basic to standard tier?

$30 ⇨ $100

End-to-end latency estimate

900 ms ⇨ 600 ms

Create a new deployment and measure performance

- Expensive
- Time consuming
- High overhead

WebPerf: “What-If” Analysis

Cloud App

Deploy with certain configuration

What if?

Service Configuration

What if I move the front-end from basic to standard tier?

$30 ⟹ $100

End-to-end latency estimate

900 ms ⟹ 600 ms

Predict performance under hypothetical configurations

- ➢ Zero cost
- ➢ Near real-time
- ➢ Zero developer effort

WebPerf: "What-If" Analysis

Cloud App

Deploy with certain configuration

+

What if?

Service Configuration

What if I move the front-end from basic to standard tier?

\$30 ⇨ \$100

=

End-to-end latency estimate

900 ms ⇨ 600 ms

Predict performance under hypothetical configurations

- ➢ Zero cost
- ➢ Near real-time
- ➢ Zero developer effort

WebPerf: Key Insights

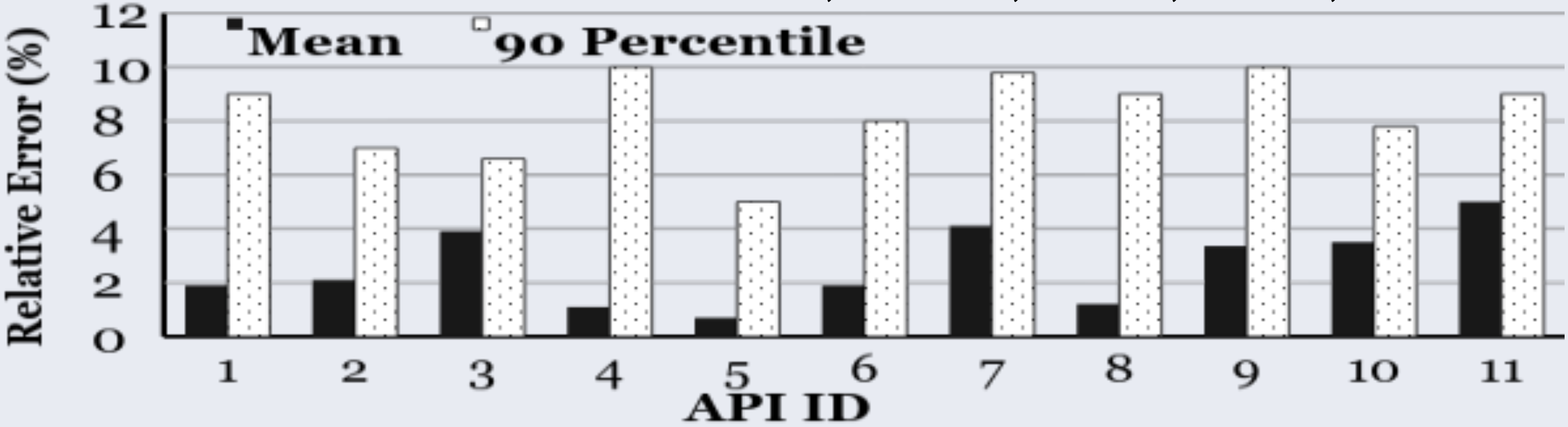
- Offline, application-independent profiling is useful
  - Modern cloud apps are built using existing services (PaaS)
  - Individual services have predictable performance
    - S3, Azure Table Storage, Dynamo DB, DocumentDB, …
  - Services are co-located inside the same datacenter
    - Tighter latency distribution

- Causal dependency within application is independent of the what-if scenarios we consider

Application-Independent Profiling

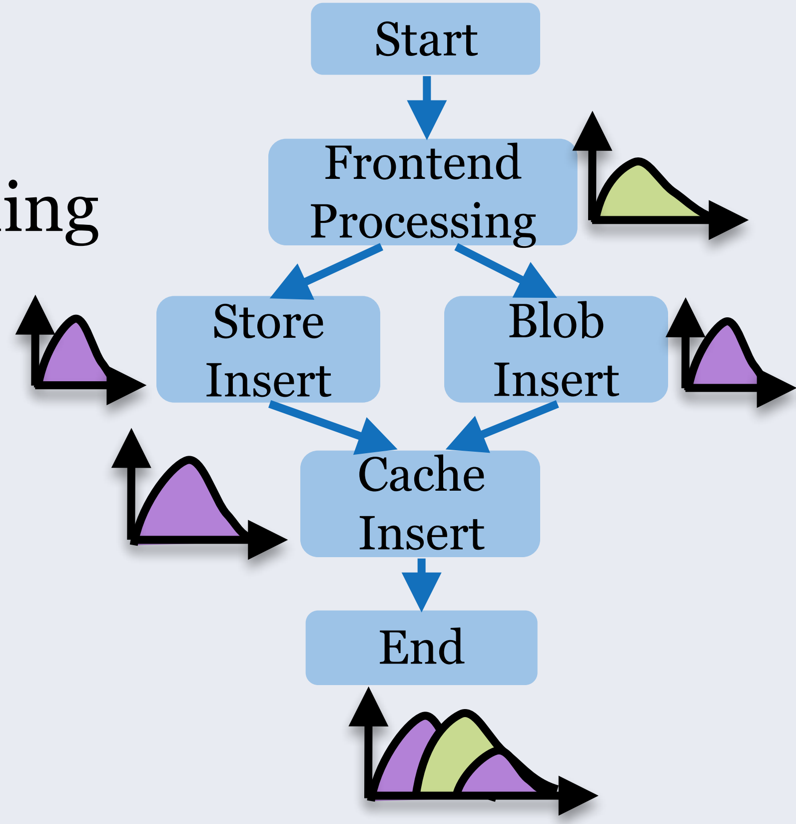
1	Delete***(Async)*** (T) ★	5	ExecuteQuerySegmented (T)	9	ToList (S)
2	UploadFromStream (B)	6	SortedSetRangeByValue (R)	10	Send (R)
3	AddMessage(Q)	7	StringGet (R)	11	ReadAsString (B)
4	Execute (T)	8	SaveChanges (S)		

★T:Table, R:Redis, S:SQL, B:Blob, Q:Queue



WebPerf Design

- Dependency graph extraction
- Application-independent profiling
- Baseline latency estimation
- Latency prediction

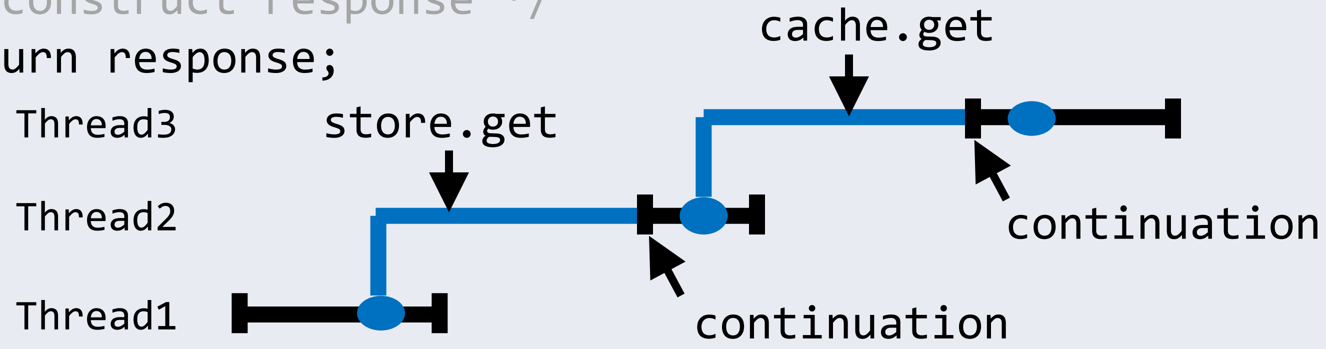


Task Asynchronous Pattern (TAP)

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```

Start task asynchronously

Continue after task finishes



Dependency Graph Extraction

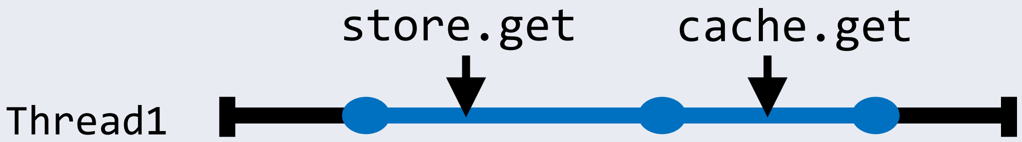
- Design Goals
  - Accurate
  - Real-time with minimal data collection
  - Zero developer effort
  - No modifications to the platform
  - Low overhead
- Automatic Binary Instrumentation
- Modern cloud applications are highly asynchronous
  - Task Asynchronous Programming Pattern

Task Asynchronous Pattern (TAP)

- Asynchronous operations with a synchronous programming pattern
- Increasingly popular for writing cloud applications
- Supported by many major languages
  - C#, Java, Python, Javascript
- Most Azure services support TAP as the ***only*** mechanism for doing asynchronous I/O
  - AWS also provides TAP APIs for .NET

Synchronous Programming

```
processRequest (input)
{
    /* process input */
    value1 = store.get(key1);
    value2 = cache.get(key2);
    /* construct response */
    return response;
}
```



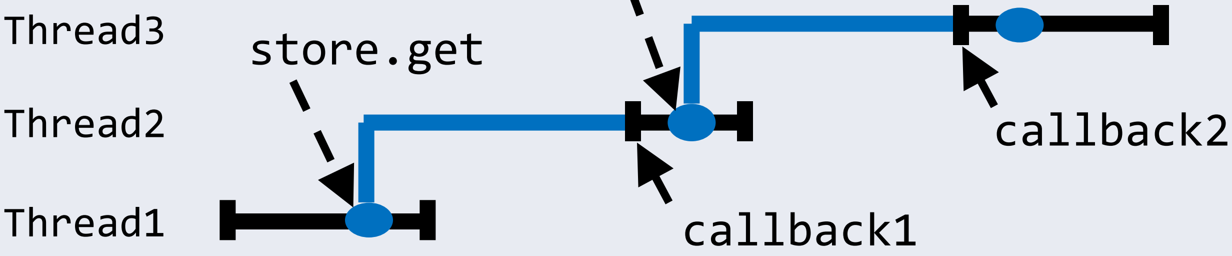
Blocking I/O limits server throughput

Asynchronous Programming Model (APM)

```
processRequest (input)
{
    /* process input */
    store.get(key1, callback1);
}

callback1 (value1)
{
    cache.get(key2, callback2);
}
```

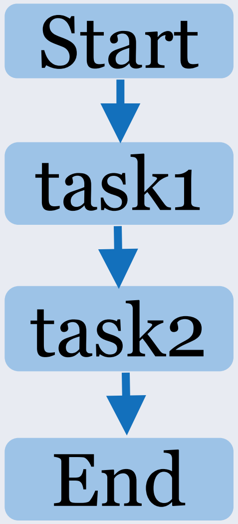
```
callback2 (value2)
{
    /* construct response */
    send(response);
}
```



Task Asynchronous Pattern (TAP)

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```

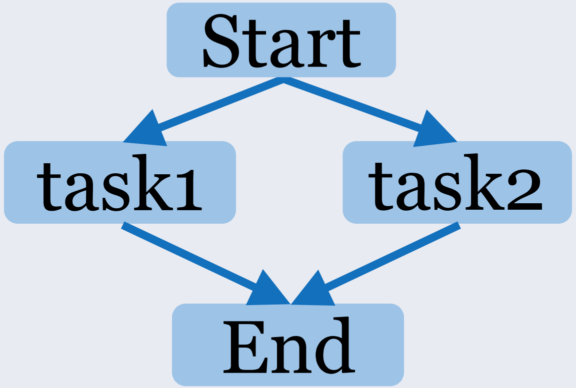
Dependency Graph



Task Asynchronous Pattern (TAP)

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    task2 = cache.get(key2);
    value1 = await task1;
    value2 = await task2;
    /* construct response */
    return response;
}
```

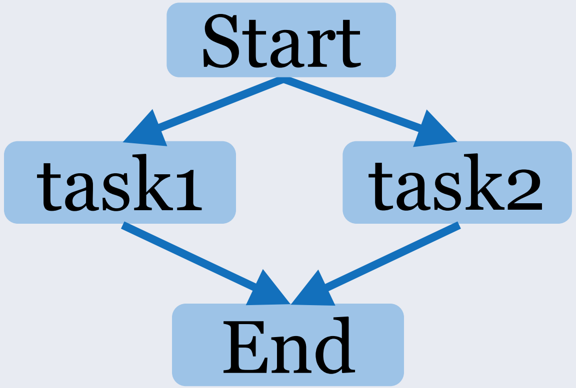
Dependency Graph



Task Asynchronous Pattern (TAP)

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    task2 = cache.get(key2);
    value1, value2 = await
            Task.WhenAll(task1, task2);
    /* construct response */
    return response;
}
```

Dependency Graph

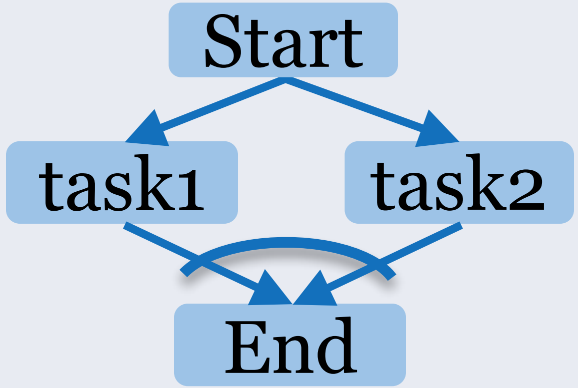


WhenAll: Continue only when all tasks finish

Task Asynchronous Pattern (TAP)

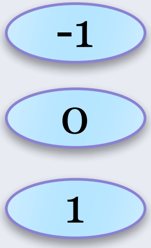
```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    task2 = cache.get(key2);
    value = await
            Task.WhenAny(task1, task2);
    /* construct response */
    return response;
}
```

Dependency Graph



WhenAny: Continue after *any one* of tasks finishes

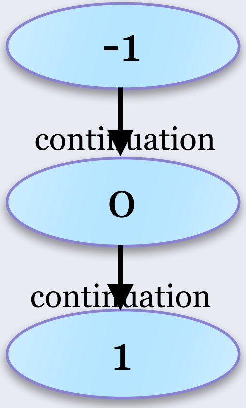
Automatic Binary Instrumentation




```
async processRequest (input)
{
  /* process input */
  task1 = store.get(key1);
  value1 = await task1;
  task2 = cache.get(key2);
  value2 = await task2;
  /* construct response */
  return response;
}
```

Instrument state machine

Track tasks and continuations




```
class processRequest__
{
  string input;
  AsyncTaskMethodBuilder builder;
  string key1, key2, response;
  int asyncId = -1;
  public void MoveNext()
  {
    asyncId = Tracker.AsyncStart(asyncId);
    Tracker.StateStart(asyncId);
    switch (state)
    {
      case -1:
        state = 0;
        /* process input */
        var task1 = store.get(key1);
        Tracker.TaskStart(task1, asyncId);
        builder.Completed(task1.Awaiter, this);
        Tracker.Await(task1, asyncId);
      case 0:
        state = 1;
        var task2 = cache.get(key1);
        Tracker.TaskStart(task2, asyncId);
        builder.Completed(task2.Awaiter, this);
        Tracker.Await(task2, asyncId);
      case 1:
        /* construct response */
```

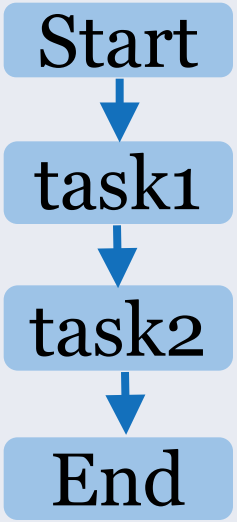
Automatic Binary Instrumentation

```
    async processRequest (input)
    {
-1      /* process input */
 0      task1 = store.get(key1);
        value1 = await task1;
 1      task2 = cache.get(key2);
        value2 = await task2;
        /* construct response */
        return response;
    }
```

Instrument state machine

Track tasks and continuations

Dependency Graph



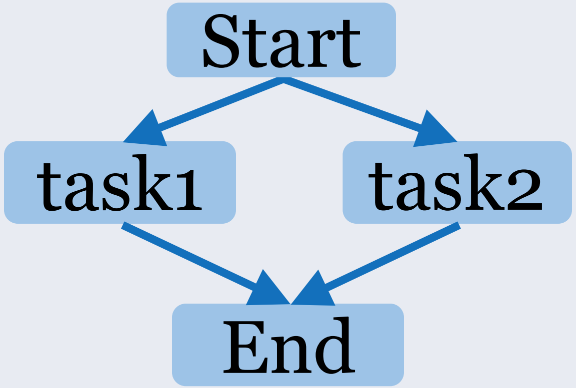
Automatic Binary Instrumentation

- **Tracking async state machines**
  - Monitor task start and completion
  - Track state machine transitions
- **Tracking pull-based continuations**
  - Link tasks to corresponding awaits
  - Link awaits to continuations
- **Tracking synchronization points**
  - Track WhenAll, WhenAny, cascaded task dependencies
- **Keeping the overhead low**
  - Instrument APIs with know signatures
  - Instrument only leaf tasks

Dependency graph extraction

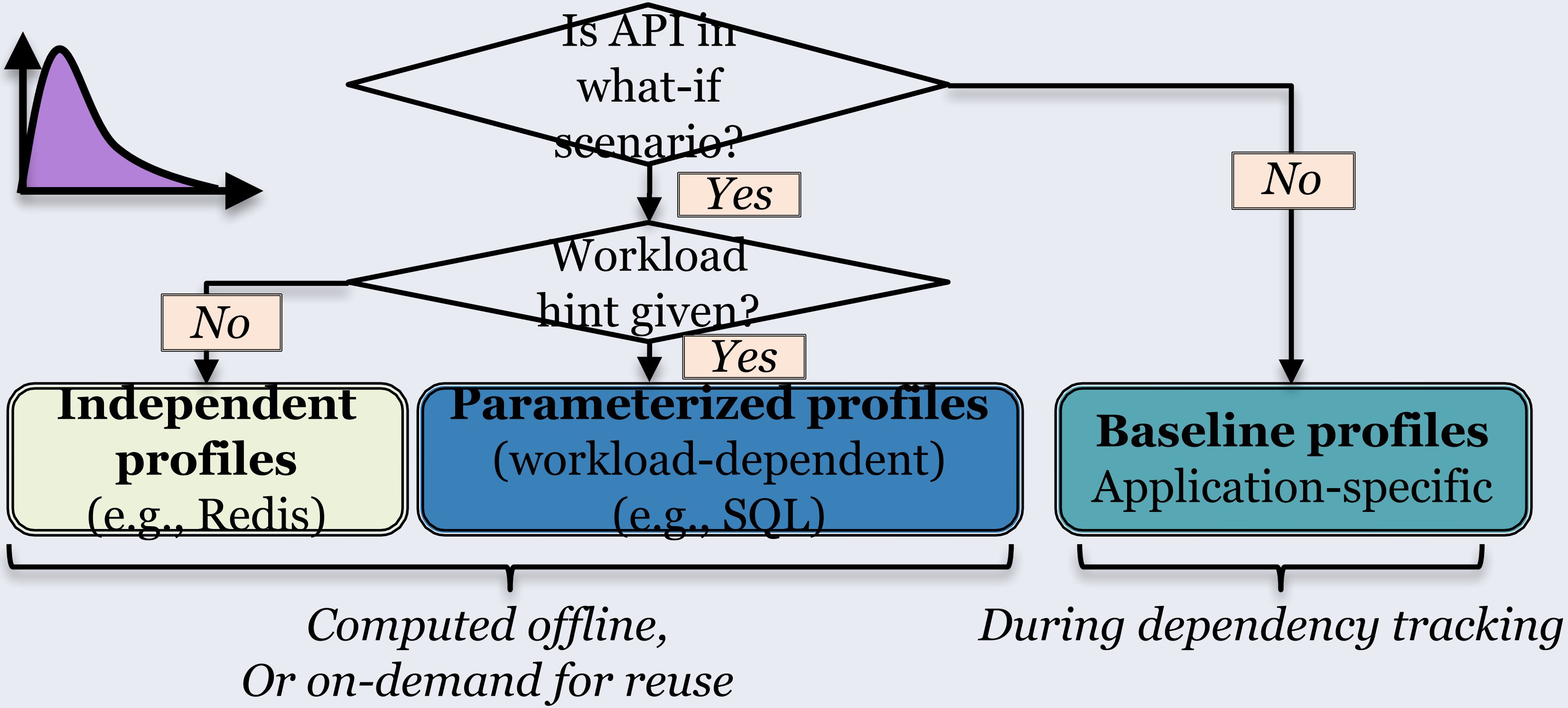
- Highly accurate
- Real-time
- Zero developer effort
- Extremely low overhead

Dependency Graph



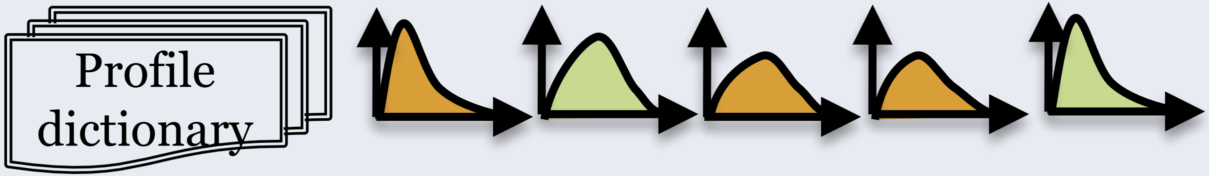
[Some Result]

API Profiling

- A profile of a cloud API is a distribution of its latency
- 

API Profiling

- WebPerf builds profiles offline and maintains in a dictionary

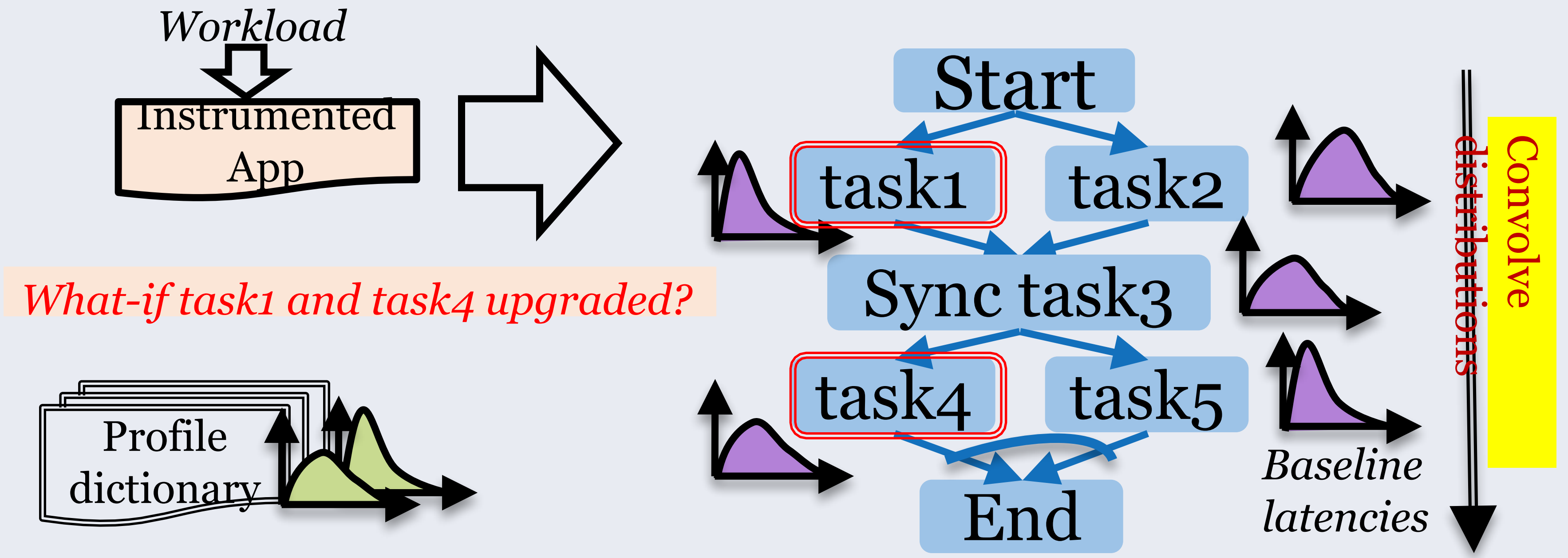


- Starts with common profiles, and builds additional profiles on-demand and reuses them

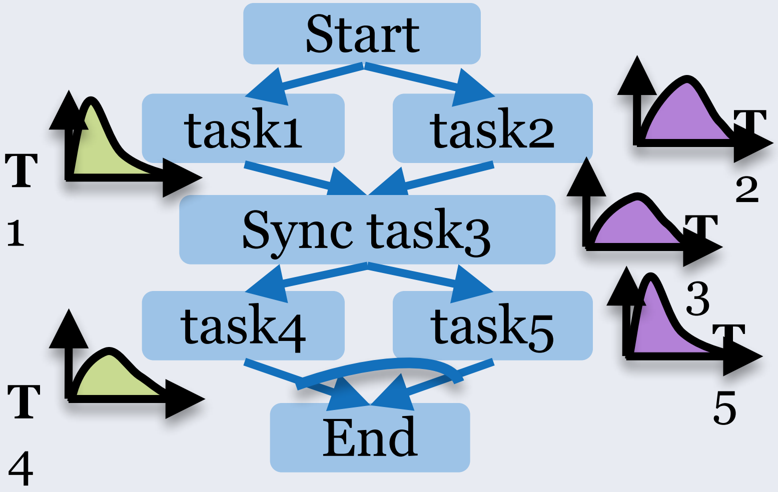
- Optimal profiling: to minimize measurement costs (details in paper)

What-If Engine

- Predicts cloud latency under a given what-if scenario



Convolving distributions



Bottom-up evaluation:

- WhenAll: ProbMax(t1, t2, ...)
- WhenAny: ProbMin(t1, t2, ...)
- WhenDone: ProbAdd(t1, t2, ...)

$T_{\{s2e\}}$= ProbAdd(ProbMax(T1, T2) , T3 , ProbMin(T4, T5))

End-to-end Latency Prediction

$$T_{e2e} = T_{Cloud} + T_{Network} + T_{Browser}$$

WebPerf

Network latency model

WebProphet[]

Combine using Monte-Carlo simulation

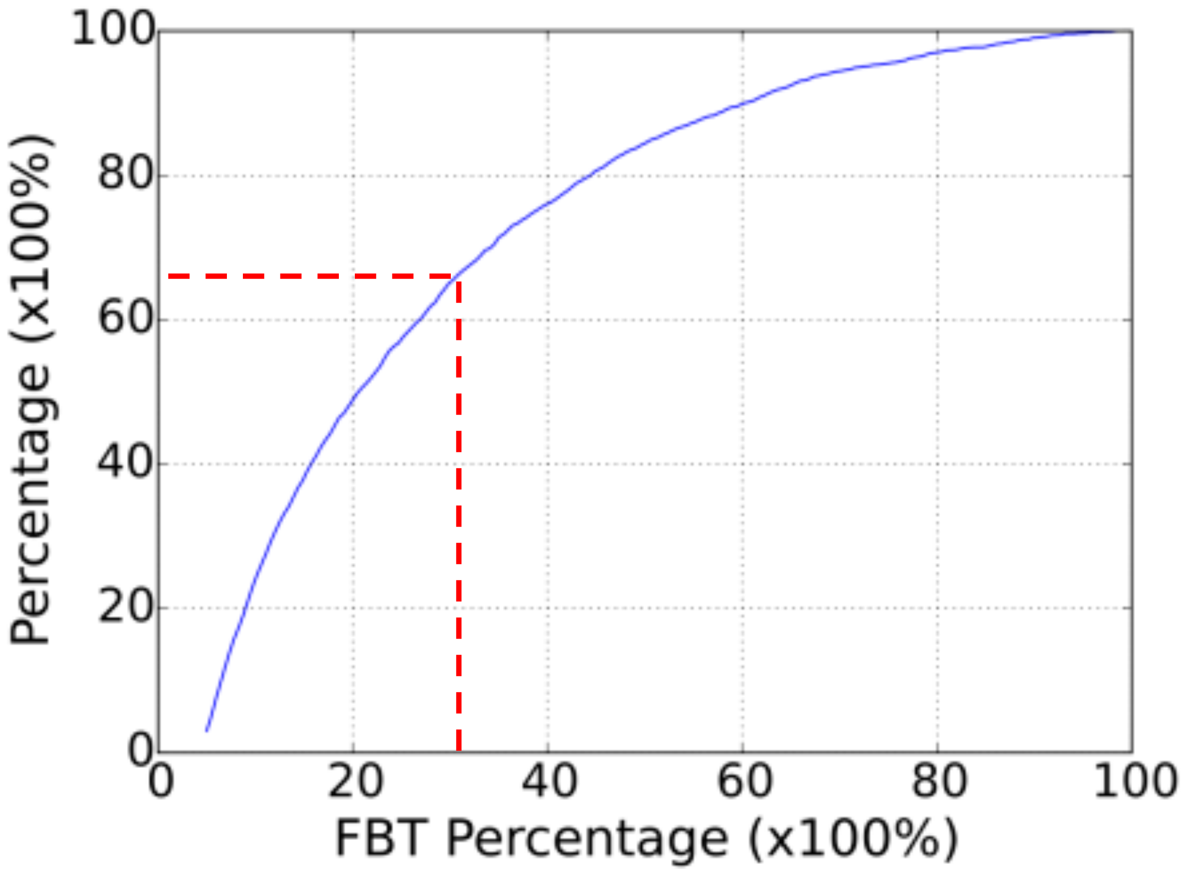
- Details in paper

WebPerf Evaluation

- Six 3rd party applications and six scenarios

Application	Azure services used	Average I/O Calls
SocialForum	Blob storage, Redis cache, Service bus, Search, Table	116
SmartStore.Net	SQL	41
ContosoAds	Blob storage, Queue, SQL, Search	56
EmailSubscriber	Blob storage, Queue, Table	26
ContactManager	Blob storage, SQL	8
CourseManager	Blob storage, SQL	44

CDF of First Byte Time Percentage

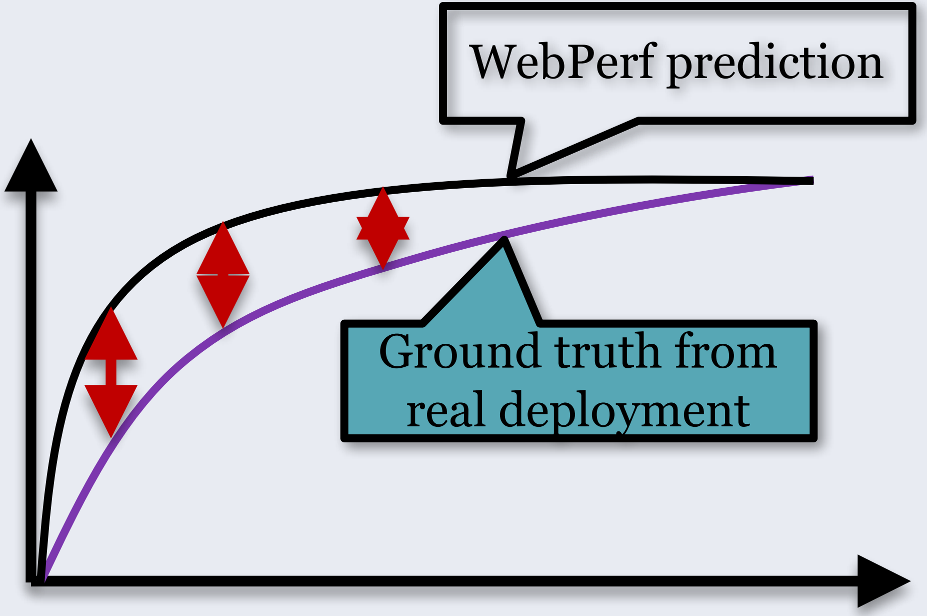


WebPerf Evaluation

- Six 3rd party applications and six scenarios

What-if scenario	Example
Tier: A resource X is upgraded to tier Y	X = A Redis cache, Y = a standard tier (from a basic tier)
Load: X concurrent requests to resource Y	X = 100 , Y = the application or a SQL database
Interference: CPU and/or memory pressure, from collocated applications, of X%	X = 50% CPU, 80% memory
Location: A resource X is deployed at location Y	X = A Redis Cache or a front end, Y = Singapore
Failure: An instance of a replicated resource X fails	X = A replicated front-end or SQL database

WebPerf Evaluation



- Metric: distribution of **relative errors**

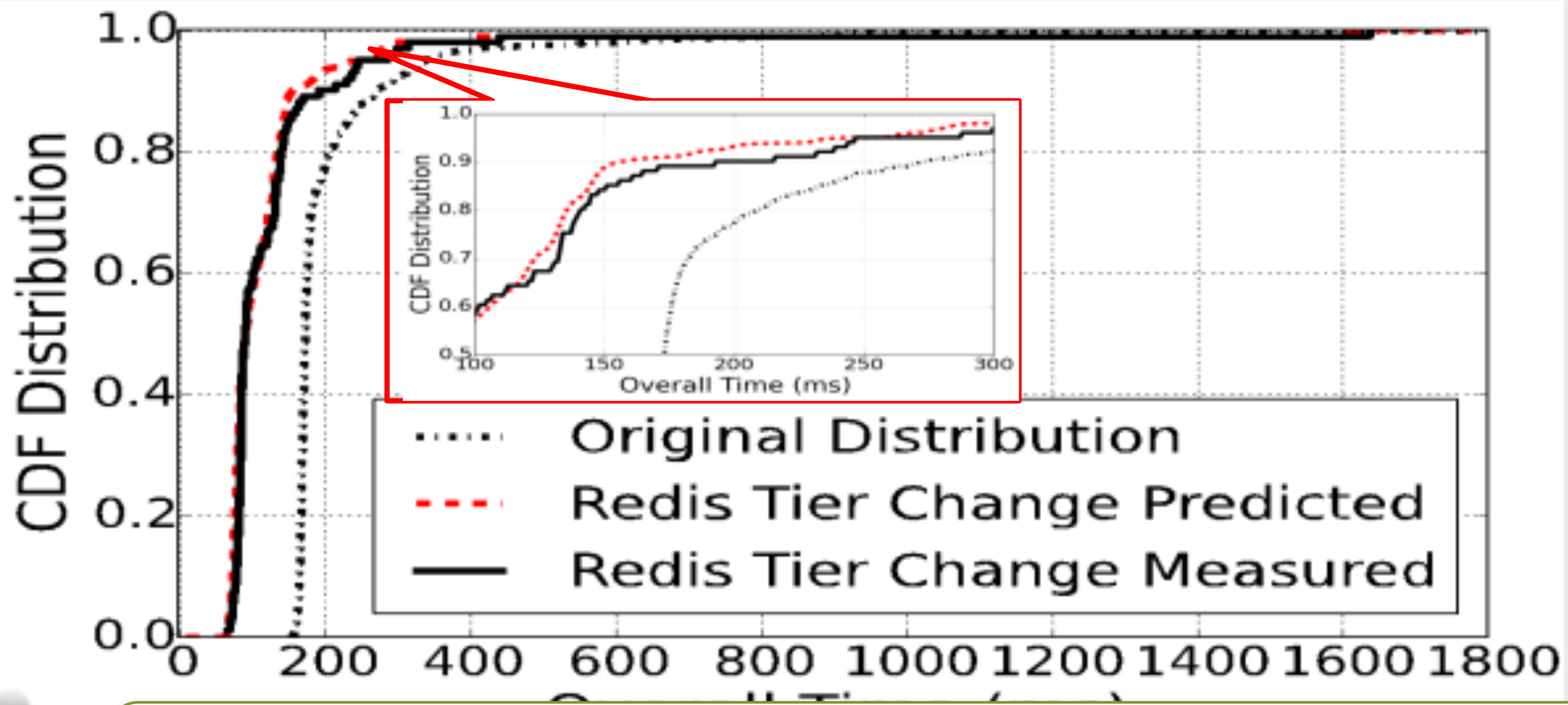
Underspecified Configuration Dimensions

Cache

Redis Cache

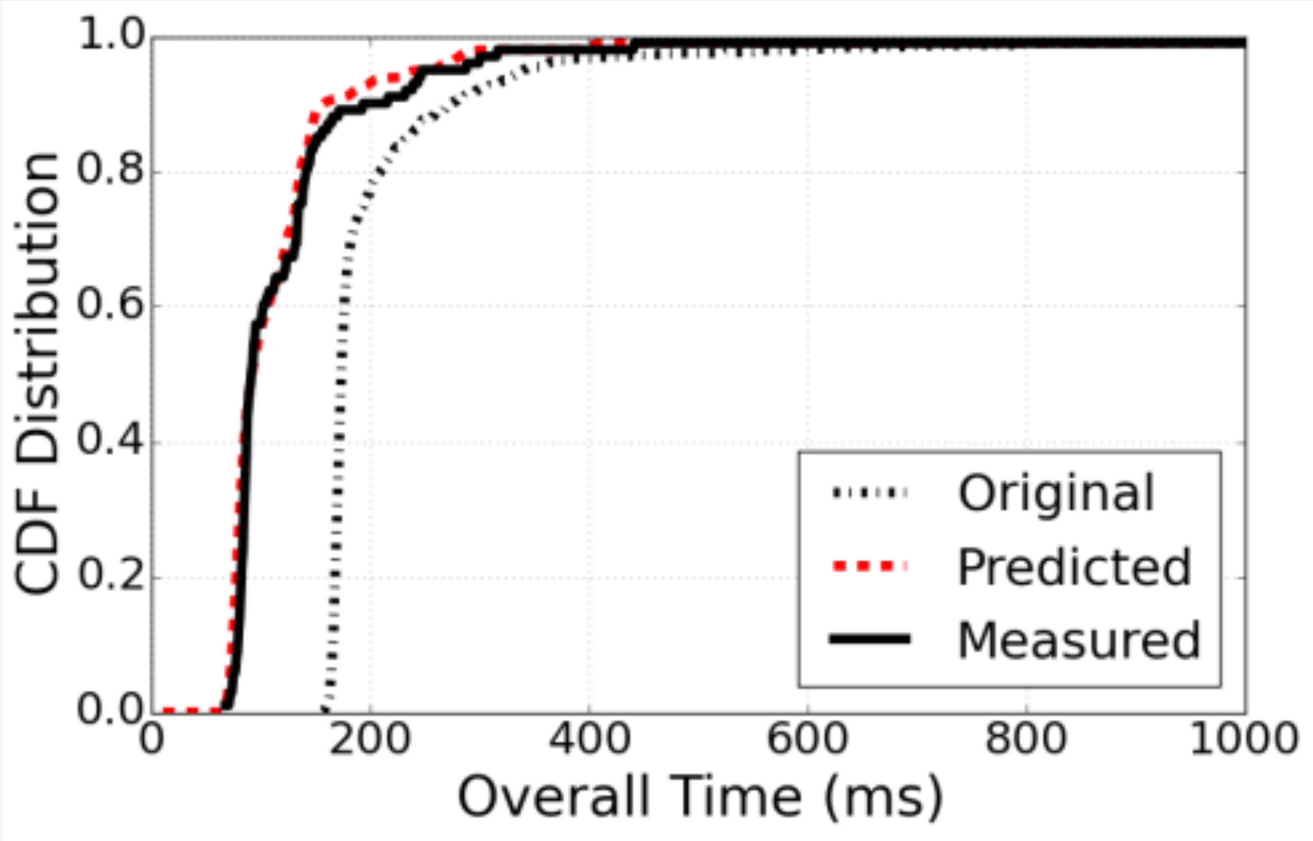
CACHE NAME	CACHE SIZE	BASIC	NETWORK PERFORMANCE	NUMBER OF CLIENT CONNECTIONS
C0	250 MB	$0.022/hr (~$16/mo)	Low	256
C1	1 GB	$0.055/hr (~$41/mo)	Low	1000
C2	2.5 GB	$0.09/hr (~$67/mo)	Moderate	2000
C3	6 GB	$0.18/hr (~$134/mo)	Moderate	5000
C4	13 GB	$0.21/hr (~$156/mo)	Moderate	10000
C5	26 GB	$0.42/hr (~$312/mo)	High	15000
C6	53 GB	$0.84/hr (~$625/mo)	Highest	20000

What-if the Redis cache is upgraded from the original standard C0 to Standard C2 tier?

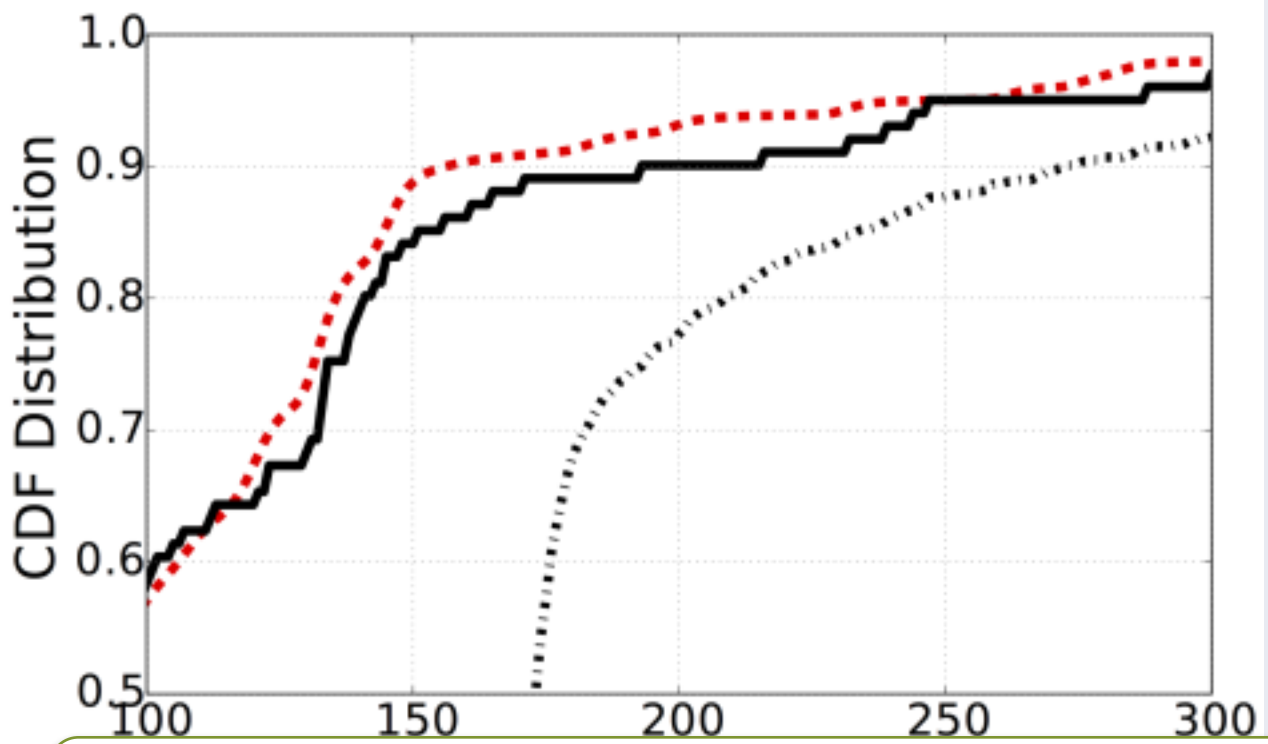


Maximum cloud side latency prediction error is only 5%

What-if the Redis cache is upgraded from the original standard C0 to Standard C2 tier?

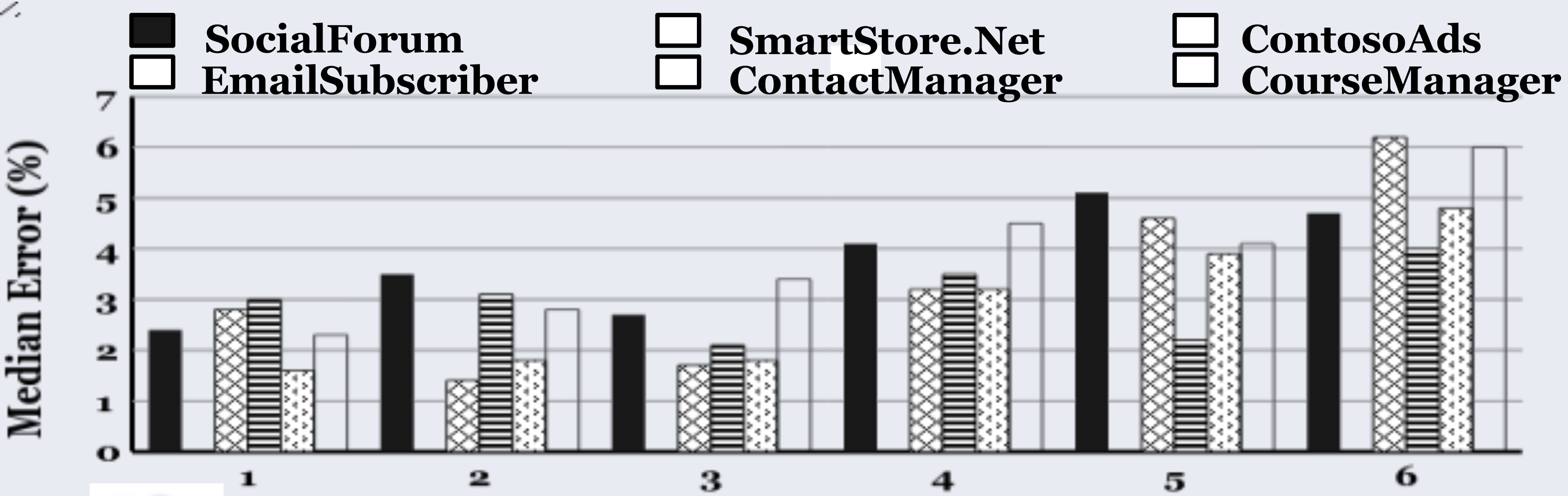


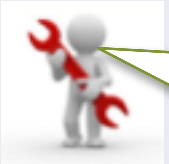
What-if the Redis cache is upgraded from the original standard C0 to Standard C2 tier?



Maximum cloud side latency prediction error is only 5%

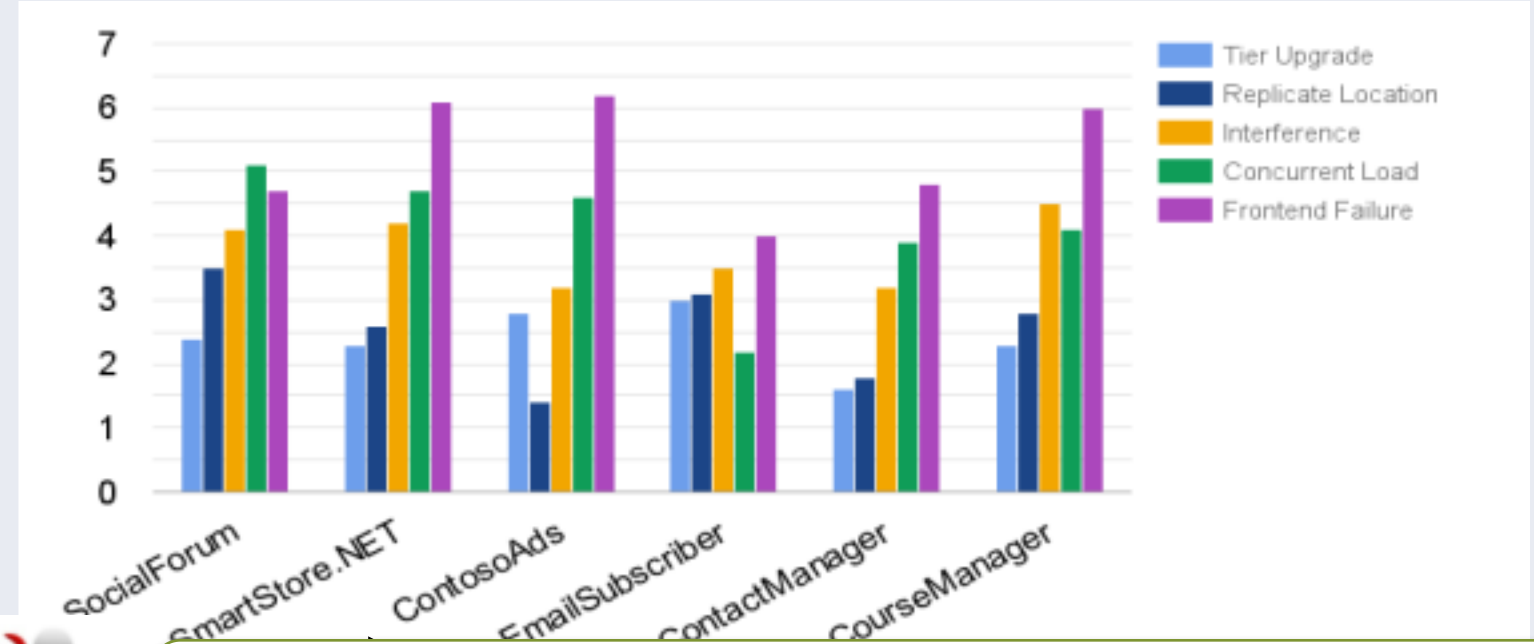
Performance for Six Applications

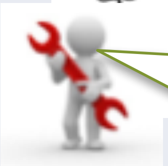




Median prediction error for cloud side latency is < 7%

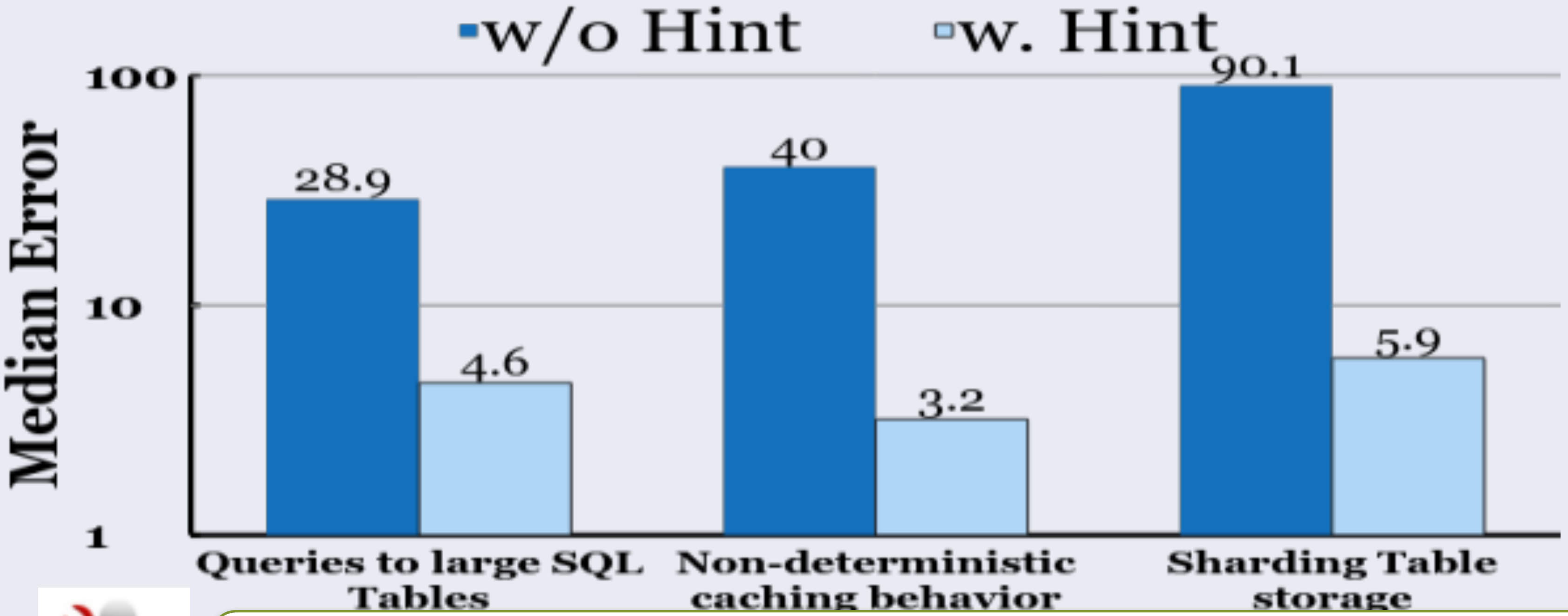
Performance for Six Applications





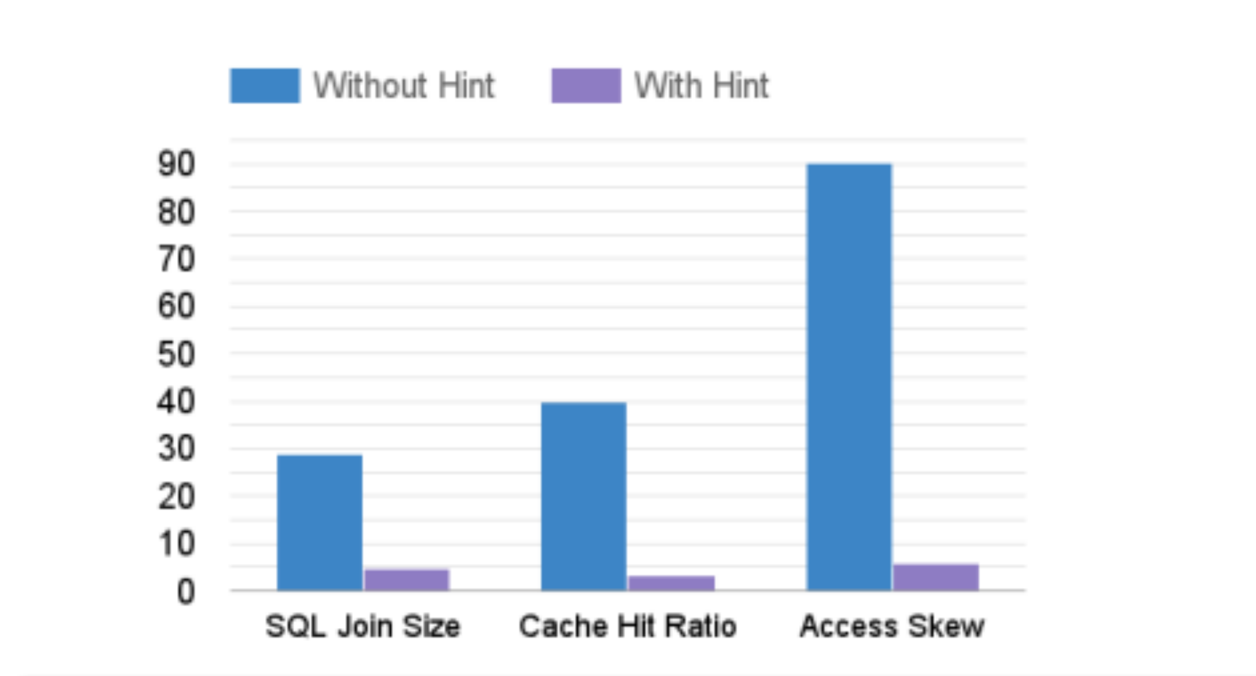
Median prediction error for cloud side latency is < 7%

Workload Hints



Workload hints can bring order of magnitude accuracy improvement

Workload Hints

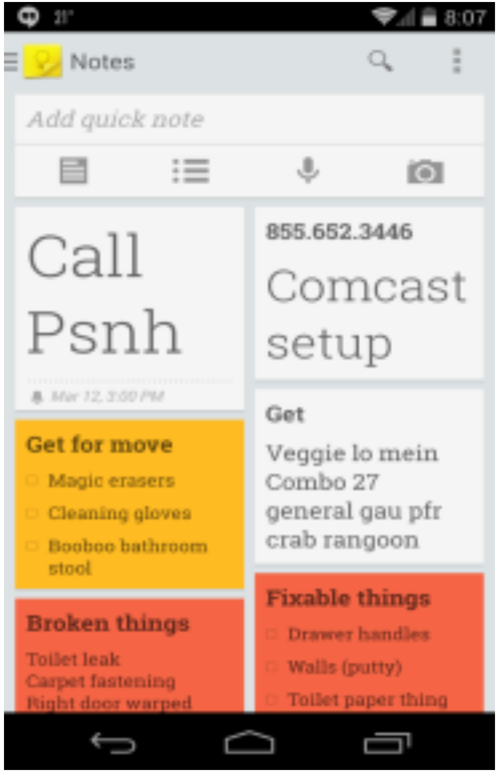


Workload hints can bring order of magnitude accuracy improvement

Other findings

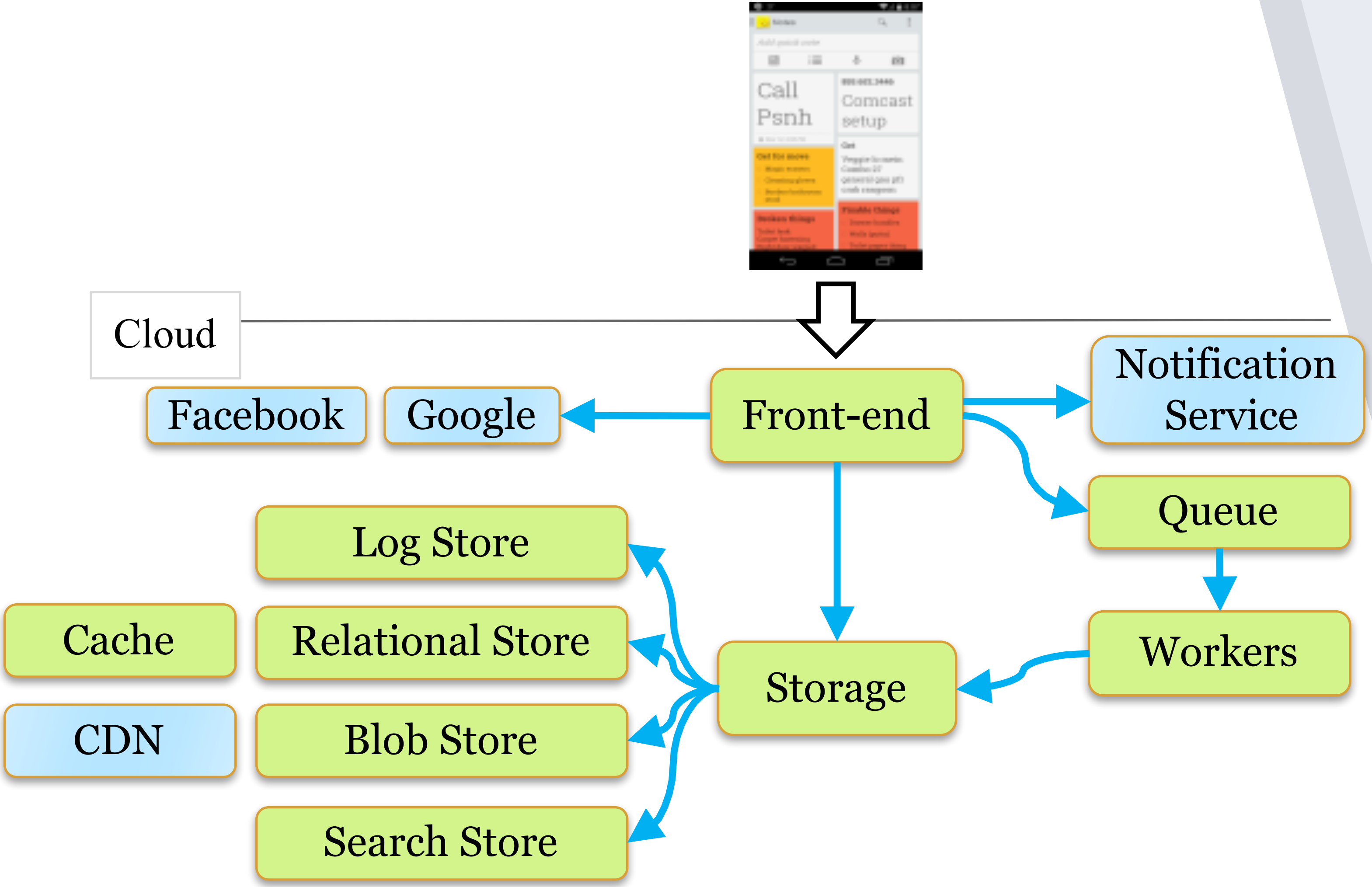
- Performance of many applications and scenarios can be predicted reasonably well
  - Thanks to cloud provider's SLAs
- Harder cases
  - Workload-dependent performance: hints help
  - High-variance profiles: prediction has high variance
  - Non-deterministic control flow (e.g., cache hit/miss):
    - Separate prediction for each control flow
  - Hard-to-profile APIs (e.g., SQL query with join)
    - Poor prediction

Modern Cloud Applications are Complex



Behind apps, several distributed, asynchronous components

Modern Cloud Applications are Complex



Hard to reason about the ~~performance~~ of cloud-hosted Web apps

Cloud-side Latency

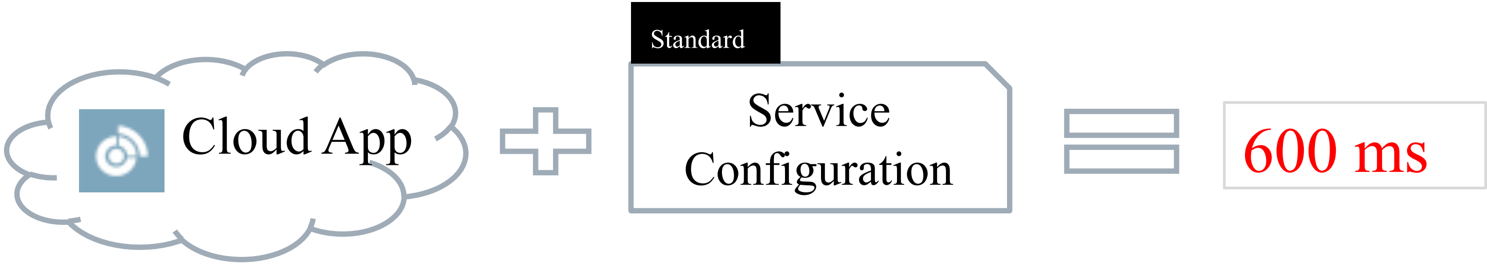
Hard to reason about the cost/performance tradeoffs of different configurations

Workload Dependence

INSTANCE	CORES	RAM	DISK SIZES	PRICE
A0	1	0.75 GB	19 GB	$0.02/hr (~$15/mo)
A1	1	1.75 GB	224 GB	$0.08/hr (~$60/mo)
A2	2	3.5 GB	489 GB	$0.16/hr (~$119/mo)
A3	4	7 GB	999 GB	$0.32/hr (~$238/mo)
A4	8	14 GB	2,039 GB	$0.64/hr (~$476/mo)

What if?

What if I move the front-end from basic to standard tier?



- Expensive
- Slow
- High Effort

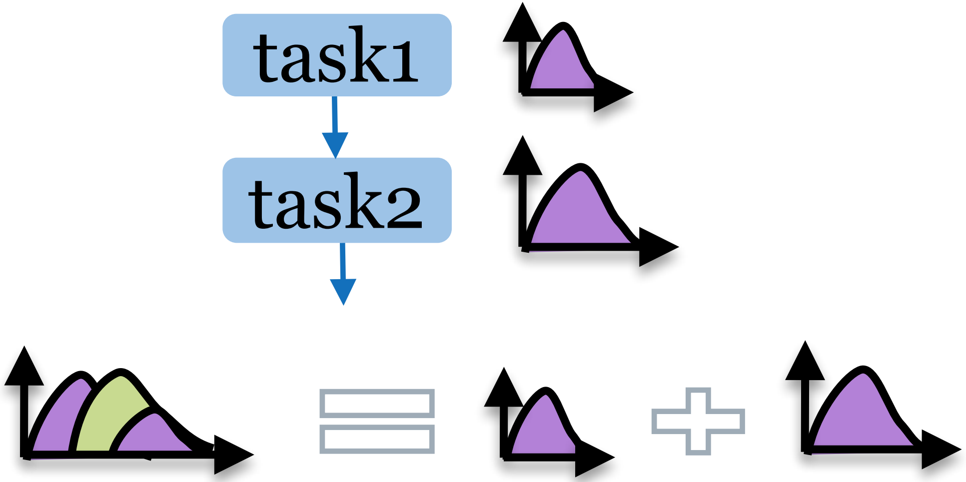
What if?

What if I move the front-end from basic to standard tier?

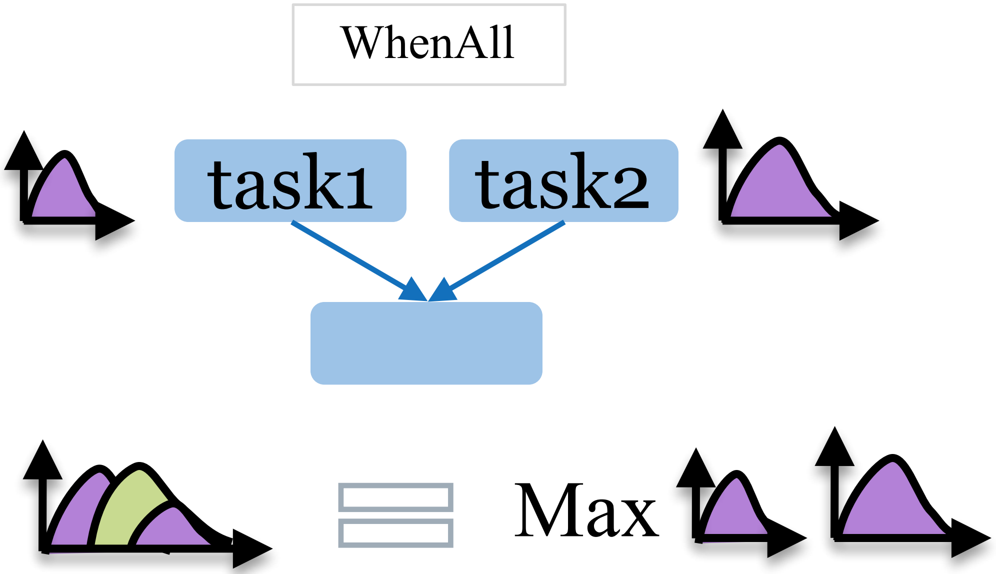
Cloud App + Service Configuration = 200 ms

+ Standard Service Configuration = 100 ms

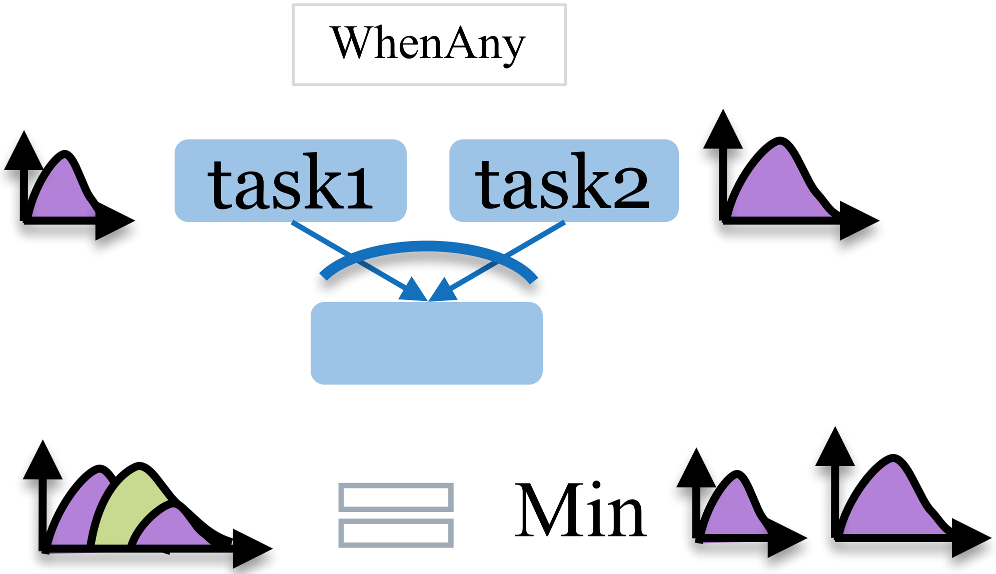
Combining latency distributions



Combining latency distributions

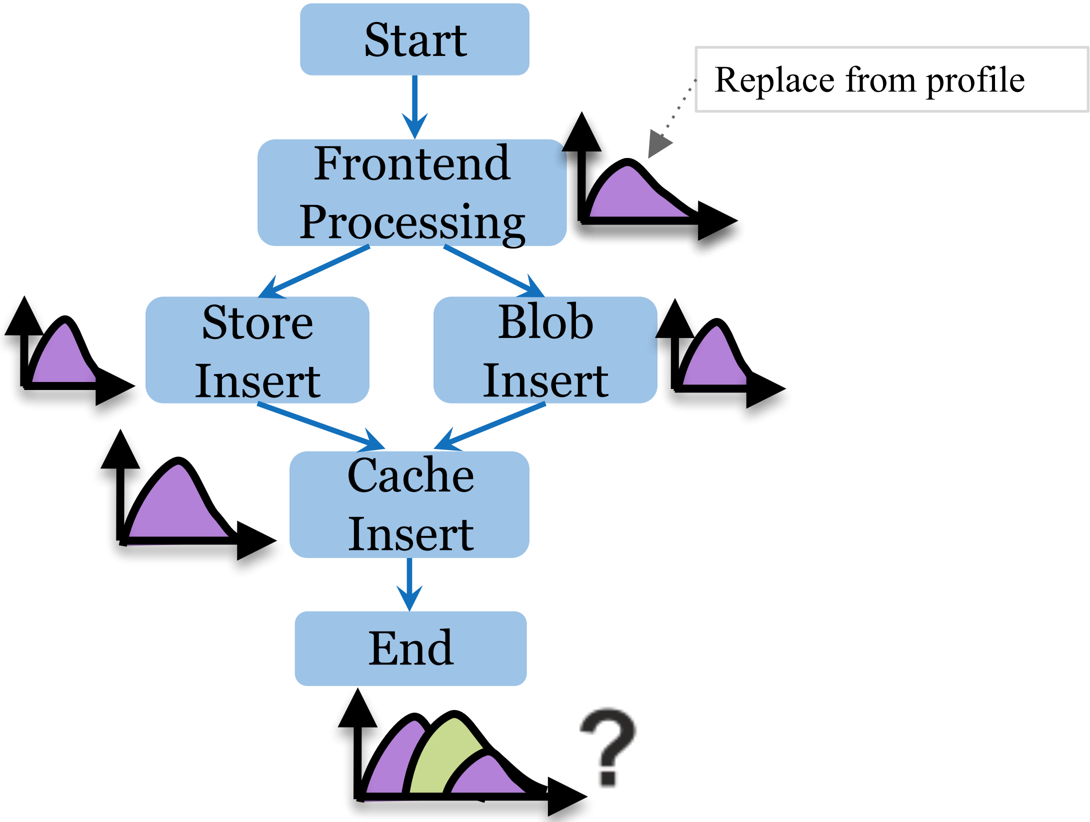


Combining latency distributions

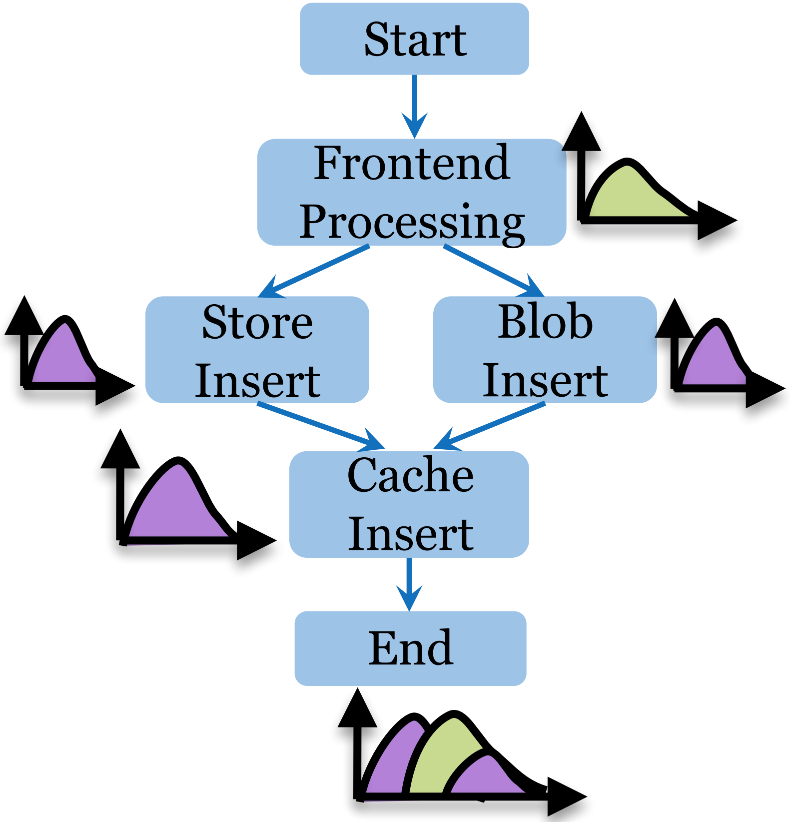


Cloud-Side Latency Estimation

What if I move the front-end from basic to standard tier?

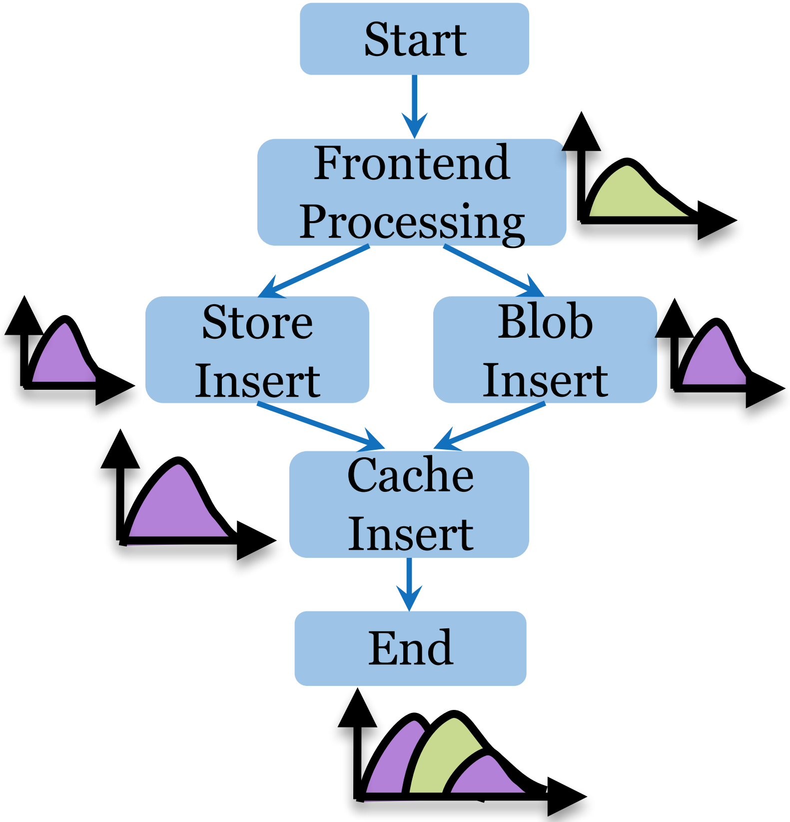


What if I move the front-end from basic to standard tier?



- Dependency graph extraction
- Baseline latency estimation
- Component Profiling
- Cloud-side latency estimation

What if I move the front-end from basic to standard tier?



Dependency graph extraction

Baseline latency estimation

Component Profiling

Cloud-side latency estimation

Developer supplies workload

Computed Offline

How accurate is WebPerf?

What is WebPerf's overhead?

What are the primary sources of prediction error?

Are workload hints necessary?

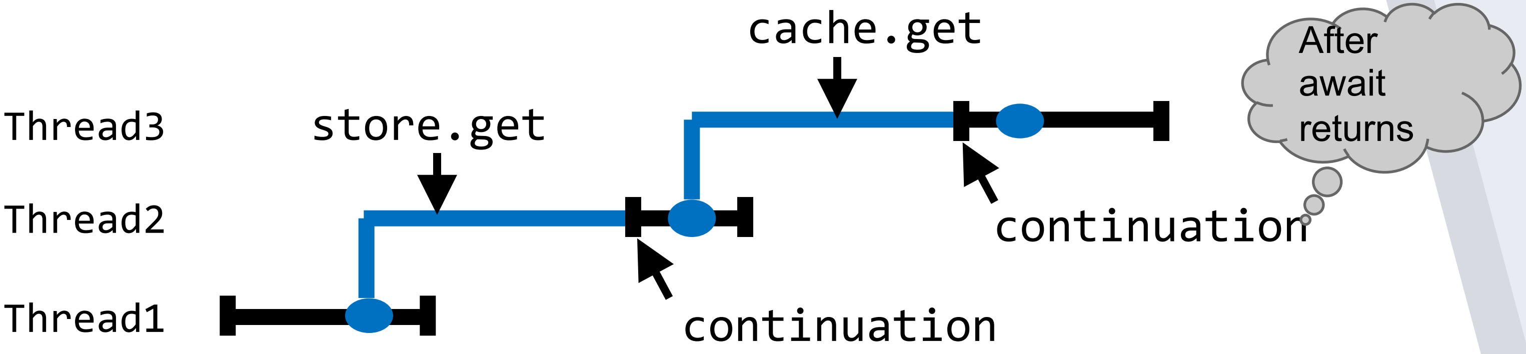
Can WebPerf predict end-to-end latency?

```
async processRequest (input)
{
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```

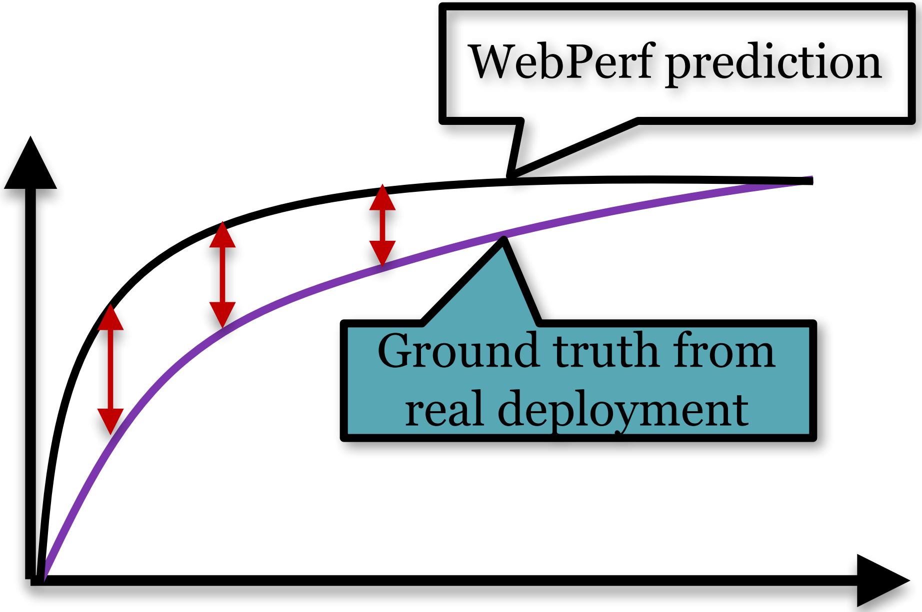
On front-end

Start task

Continue



Distributional Difference



Measure of Prediction Accuracy