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

Yurong Jiang
Lenin Ravindranath
Suman Nath
Ramesh Govindan
A Cloud-Hosted Web Application
Modern Cloud Applications are Complex
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

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

Each choice impacts latency
<table>
<thead>
<tr>
<th>Premium</th>
<th>DTUs ²</th>
<th>MAX STORAGE PER DB</th>
<th>PRICE ³</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>125</td>
<td>500 GB</td>
<td>~$465/mo</td>
</tr>
<tr>
<td>P2</td>
<td>250</td>
<td>500 GB</td>
<td>~$930/mo</td>
</tr>
<tr>
<td>P4</td>
<td>500</td>
<td>500 GB</td>
<td>~$1,860/mo</td>
</tr>
<tr>
<td>P6</td>
<td>1000</td>
<td>500 GB</td>
<td>~$3,720/mo</td>
</tr>
<tr>
<td>P11</td>
<td>1750</td>
<td>1 TB</td>
<td>~$7,001/mo</td>
</tr>
</tbody>
</table>

Latency implications hard to understand
Exploring Configuration Choices

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
Cloud deployments well-engineered
- Components designed for predictable latency
- Often co-located in same datacenter
Many component profiles are application-independent

The dependency graph is usually independent of what-if scenario
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
Task Asynchronous Programming

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

Prior work has not considered this
async processRequest (input) {
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}
```javascript
async processRequest (input) {
  /* process input */
  task1 = store.get(key1);
  value1 = await task1;
  task2 = cache.get(key2);
  value2 = await task2;
  /* construct response */
  return response;
}
```
Asynchronous Parallel Operations

**WhenAll**: Continue only when all tasks finish

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

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

.NET compiler generates this
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*.
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

- **Cache**
  - Redis Cache

- **Relational Store**
  - Sharded TableStore

- **Size**
- **Hit rate**
- **Skew**
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?

Cloud-Side Latency Estimation
Simple operations on distributions suffice
WebPerf is accurate, fast, cheap, and requires low developer effort

❖ How accurate is WebPerf?
❖ Are workload hints necessary?
<table>
<thead>
<tr>
<th>Application</th>
<th>Azure components used</th>
<th>Average I/O Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocialForum</td>
<td>Blob storage, Redis cache, Service bus, Search, Table</td>
<td>116</td>
</tr>
<tr>
<td>SmartStore.Net</td>
<td>SQL</td>
<td>41</td>
</tr>
<tr>
<td>ContosoAds</td>
<td>Blob storage, Queue, SQL, Search</td>
<td>56</td>
</tr>
<tr>
<td>EmailSubscriber</td>
<td>Blob storage, Queue, Table</td>
<td>26</td>
</tr>
<tr>
<td>ContactManager</td>
<td>Blob storage, SQL</td>
<td>8</td>
</tr>
<tr>
<td>CourseManager</td>
<td>Blob storage, SQL</td>
<td>44</td>
</tr>
<tr>
<td>What-if scenario</td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Tier</strong>: A component X is upgraded to tier Y</td>
<td>X = A Redis cache, Y = a standard tier (from a basic tier)</td>
<td></td>
</tr>
<tr>
<td><strong>Load</strong>: X concurrent requests to component Y</td>
<td>X = 100, Y = the application or a SQL database</td>
<td></td>
</tr>
<tr>
<td><strong>Interference</strong>: CPU and/or memory pressure, from collocated applications, of X%</td>
<td>X = 50% CPU, 80% memory</td>
<td></td>
</tr>
<tr>
<td><strong>Location</strong>: A component X is deployed at location Y</td>
<td>X = A Redis Cache or a front end, Y = Singapore</td>
<td></td>
</tr>
<tr>
<td><strong>Failure</strong>: An instance of a replicated component X fails</td>
<td>X = A replicated front-end or SQL database</td>
<td></td>
</tr>
</tbody>
</table>
What if I move the Redis cache in SocialForum from basic to standard tier?

Configuration choices can significantly impact latency.
What if I move the Redis cache from basic to standard tier?

Prediction closely matches ground truth
Median prediction error under 7%

Difference between predicted distribution and ground truth
Accuracy

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

- Facebook
- Google
- Front-end
- Notification Service
- Log Store
- Relational Store
- Cache
- Blob Store
- CDN
- Search Store
- Queue
- Workers
Large number of configuration choices

Facebook

Google

Front-end

Notification Service

Queue

<table>
<thead>
<tr>
<th>INSTANCE</th>
<th>CORES</th>
<th>RAM</th>
<th>DISK SIZE</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>1</td>
<td>0.75 GB</td>
<td>19 GB</td>
<td>$0.02/hr (~$15/mo)</td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>1.75 GB</td>
<td>224 GB</td>
<td>$0.08/hr (~$60/mo)</td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td>3.5 GB</td>
<td>489 GB</td>
<td>$0.16/hr (~$119/mo)</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>7 GB</td>
<td>999 GB</td>
<td>$0.32/hr (~$228/mo)</td>
</tr>
<tr>
<td>A4</td>
<td>8</td>
<td>14 GB</td>
<td>2,039 GB</td>
<td>$0.64/hr (~$478/mo)</td>
</tr>
<tr>
<td>A5</td>
<td>2</td>
<td>14 GB</td>
<td>489 GB</td>
<td>$0.35/hr (~$260/mo)</td>
</tr>
<tr>
<td>A6</td>
<td>4</td>
<td>28 GB</td>
<td>999 GB</td>
<td>$0.71/hr (~$528/mo)</td>
</tr>
<tr>
<td>A7</td>
<td>8</td>
<td>56 GB</td>
<td>2,039 GB</td>
<td>$1.41/hr (~$1,045/mo)</td>
</tr>
</tbody>
</table>
Large number of configuration choices

- Facebook
- Google
- Log Store
- Relational Store
- Cache
- Blob Store
- CDN
- Search Store

### Azure SQL

**Basic**

- **DTUs**: 5
- **Max Storage Per DB**: 2 GB
- **Price**: $5/mo

**Standard**

- **S0**: 10 DTUs, 250 GB storage, $15/mo
- **S1**: 20 DTUs, 250 GB storage, $30/mo
- **S2**: 50 DTUs, 250 GB storage, $75/mo
- **S3**: 100 DTUs, 250 GB storage, $150/mo

**Premium**

- **P1**: 125 DTUs, 500 GB storage, $465/mo
- **P2**: 250 DTUs, 500 GB storage, $930/mo
- **P4**: 500 DTUs, 500 GB storage, $1,860/mo
- **P6**: 1000 DTUs, 500 GB storage, $3,720/mo
- **P11**: 1750 DTUs, 1 TB, $7,001/mo

1. Price is based on usage and varies with DTUs and storage requirements.
Large number of configuration choices

<table>
<thead>
<tr>
<th>Cache Name</th>
<th>Cache Size</th>
<th>Basic Price</th>
<th>Network Performance</th>
<th>Number of Client Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>250 MB</td>
<td>$0.02/MB/hr</td>
<td>Low</td>
<td>256</td>
</tr>
<tr>
<td>C1</td>
<td>1 GB</td>
<td>$0.05/MB/hr</td>
<td>Low</td>
<td>1000</td>
</tr>
<tr>
<td>C2</td>
<td>2.5 GB</td>
<td>$0.09/MB/hr</td>
<td>Moderate</td>
<td>2000</td>
</tr>
<tr>
<td>C3</td>
<td>6 GB</td>
<td>$0.16/MB/hr</td>
<td>Moderate</td>
<td>5000</td>
</tr>
<tr>
<td>C4</td>
<td>13 GB</td>
<td>$0.31/MB/hr</td>
<td>Moderate</td>
<td>10000</td>
</tr>
<tr>
<td>C5</td>
<td>26 GB</td>
<td>$0.42/MB/hr</td>
<td>High</td>
<td>15000</td>
</tr>
<tr>
<td>C6</td>
<td>53 GB</td>
<td>$0.84/MB/hr</td>
<td>Highest</td>
<td>20000</td>
</tr>
<tr>
<td>P2</td>
<td>13 GB</td>
<td>$1.11/MB/hr</td>
<td>Moderate</td>
<td>15000</td>
</tr>
<tr>
<td>P3</td>
<td>26 GB</td>
<td>$2.16/MB/hr</td>
<td>High</td>
<td>30000</td>
</tr>
</tbody>
</table>

Services:
- Facebook
- Google
- Log Store
- Cache
- Relational Store
- CDN
- Blob Store
- Search Store
- Redis Cache
Large number of configuration choices

Reasoning about cost-performance trade-off is hard!
Cost-Performance Trade-off

- Configuration does not directly map to performance
- End-to-end latency depends on application’s causal dependency

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cores</th>
<th>RAM</th>
<th>Disk Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>1</td>
<td>0.75 GB</td>
<td>19 GB</td>
<td>$0.02/hr (≈$117/mon)</td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>1.75 GB</td>
<td>224 GB</td>
<td>$0.08/hr (≈$60/mon)</td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td>3.5 GB</td>
<td>489 GB</td>
<td>$0.16/hr (≈$1,115/mon)</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>7 GB</td>
<td>999 GB</td>
<td>$0.32/hr (≈$2,236/mon)</td>
</tr>
<tr>
<td>A4</td>
<td>8</td>
<td>14 GB</td>
<td>2,039 GB</td>
<td>$0.64/hr (≈$4,710/mon)</td>
</tr>
</tbody>
</table>

Start → Store Insert (100 ms) → Cache Insert (20 ms) → End (100 ms)

End-to-end latency depends on application’s causal dependency.
“What-If” Analysis

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

What if?

Service Configuration

End-to-end latency estimate

Create a new deployment and measure performance

➢ Expensive
➢ Time consuming
➢ High overhead
WebPerf: “What-If” Analysis

Deploy with certain configuration

Cloud App

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

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

$30 \rightarrow $100

900 ms \rightarrow 600 ms

End-to-end latency estimate

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

<table>
<thead>
<tr>
<th>API ID</th>
<th>Operation</th>
<th>Data Store(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Delete(Async)</strong> (T)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>UploadFromStream (B)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AddMessage(Q)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Execute (T)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ExecuteQuerySegmented (T)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SortedSetRangeByValue (R)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>StringGet (R)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SaveChanges (S)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ToList (S)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Send (R)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ReadAsString (B)</td>
<td></td>
</tr>
</tbody>
</table>

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

![Bar Chart](chart.png)

- Mean
- 90 Percentile

<table>
<thead>
<tr>
<th>API ID</th>
<th>Relative Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>
WebPerf Design

- Dependency graph extraction
- Application-independent profiling
- Baseline latency estimation
- Latency prediction
async processRequest (input) {
    /* process input */
    task1 = store.get(key1);
    value1 = await task1;
    task2 = cache.get(key2);
    value2 = await task2;
    /* construct response */
    return response;
}

Thread1

Thread2

Thread3

Start task asynchronously

Continue after task finishes

cache.get

continuation

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

*WhenAny*: Continue after *any one* of tasks finishes
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);
                break;
            case 0:
                state = 1;
                var task2 = cache.get(key2);
                Tracker.TaskStart(task2, asyncId);
                builder.Completed(task2.Awaiter, this);
                Tracker.Await(task2, asyncId);
                break;
            case 1:
                /* construct response */
                builder.SetResult(response);
                Tracker.StateEnd(asyncId);
                Tracker.AsyncEnd(asyncId);
                return;
        }
        Tracker.StateEnd(asyncId);
    }
}
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

Dependency Graph

Start
→ task1
→ task2
→ End
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 known signatures
  - Instrument only leaf tasks
Dependency graph extraction

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

[Some Result]
A profile of a cloud API is a distribution of its latency.

- **Parameterized profiles** (workload-dependent) (e.g., SQL)
  - Computed offline, or on-demand for reuse
  - During dependency tracking

- **Baseline profiles** (Application-specific)
  - Computed offline, or on-demand for reuse

- **Independent profiles** (e.g., Redis)

- Is API in what-if scenario?
  - Yes
  - Workload hint given?
    - Yes
    - Parameterized profiles (workload-dependent) (e.g., SQL)
    - No
    - Independent profiles (e.g., Redis)
  - No

<table>
<thead>
<tr>
<th>Yes/No</th>
<th>Workload hint given?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
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

What-if task1 and task4 upgraded?

Workload

Instrumented App

Profile dictionary

Baseline latencies

Start

Sync task3

task1

End

task2

Profile dictionary

Convolve distributions
Convolving distributions

Bottom-up evaluation:

- When **All**: \( \text{ProbMax}(t_1, t_2, ...) \)
- When **Any**: \( \text{ProbMin}(t_1, t_2, ...) \)
- When **Done**: \( \text{ProbAdd}(t_1, t_2, ...) \)

\[
T_{s2e} = \text{ProbAdd}\left(\text{ProbMax}(T_1, T_2) , T_3 , \text{ProbMin}(T_4, T_5) \right)
\]
End-to-end Latency Prediction

\[ T_{e2e} = T_{\text{Cloud}} + T_{\text{Network}} + T_{\text{Browser}} \]

- WebPerf
- Network latency model
- WebProphet

Combine using Monte-Carlo simulation

- Details in paper
**WebPerf Evaluation**

- **Six 3<sup>rd</sup> party applications and six scenarios**

<table>
<thead>
<tr>
<th>Application</th>
<th>Azure services used</th>
<th>Average I/O Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocialForum</td>
<td>Blob storage, Redis cache, Service bus, Search, Table</td>
<td>116</td>
</tr>
<tr>
<td>SmartStore.Net</td>
<td>SQL</td>
<td>41</td>
</tr>
<tr>
<td>ContosoAds</td>
<td>Blob storage, Queue, SQL, Search</td>
<td>56</td>
</tr>
<tr>
<td>EmailSubscriber</td>
<td>Blob storage, Queue, Table</td>
<td>26</td>
</tr>
<tr>
<td>ContactManager</td>
<td>Blob storage, SQL</td>
<td>8</td>
</tr>
<tr>
<td>CourseManager</td>
<td>Blob storage, SQL</td>
<td>44</td>
</tr>
</tbody>
</table>
CDF of First Byte Time Percentage
WebPerf Evaluation

- Six 3rd party applications and **six scenarios**

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

- Metric: distribution of relative errors

Ground truth from real deployment

WebPerf prediction
<table>
<thead>
<tr>
<th>Cache Name</th>
<th>Cache Size</th>
<th>Basic</th>
<th>Network Performance</th>
<th>Number of Client Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>250 MB</td>
<td>$0.022/hr (~$16/mo)</td>
<td>Low</td>
<td>256</td>
</tr>
<tr>
<td>C1</td>
<td>1 GB</td>
<td>$0.055/hr (~$41/mo)</td>
<td>Low</td>
<td>1000</td>
</tr>
<tr>
<td>C2</td>
<td>2.5 GB</td>
<td>$0.09/hr (~$67/mo)</td>
<td>Moderate</td>
<td>2000</td>
</tr>
<tr>
<td>C3</td>
<td>6 GB</td>
<td>$0.18/hr (~$134/mo)</td>
<td>Moderate</td>
<td>5000</td>
</tr>
<tr>
<td>C4</td>
<td>13 GB</td>
<td>$0.21/hr (~$156/mo)</td>
<td>Moderate</td>
<td>10000</td>
</tr>
<tr>
<td>C5</td>
<td>26 GB</td>
<td>$0.42/hr (~$312/mo)</td>
<td>High</td>
<td>15000</td>
</tr>
<tr>
<td>C6</td>
<td>53 GB</td>
<td>$0.84/hr (~$625/mo)</td>
<td>Highest</td>
<td>20000</td>
</tr>
</tbody>
</table>

Underspecified Configuration Dimensions
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 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
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
<table>
<thead>
<tr>
<th>INSTANCE</th>
<th>CORES</th>
<th>RAM</th>
<th>DISK SIZES</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>1</td>
<td>0.75 GB</td>
<td>19 GB</td>
<td>$0.02/hr (~$15/yr)</td>
</tr>
<tr>
<td>A1</td>
<td>1</td>
<td>1.75 GB</td>
<td>224 GB</td>
<td>$0.08/hr (~$60/yr)</td>
</tr>
<tr>
<td>A2</td>
<td>2</td>
<td>3.5 GB</td>
<td>409 GB</td>
<td>$0.16/hr (~$115/yr)</td>
</tr>
<tr>
<td>A3</td>
<td>4</td>
<td>7 GB</td>
<td>999 GB</td>
<td>$0.32/hr (~$238/yr)</td>
</tr>
<tr>
<td>A4</td>
<td>8</td>
<td>14 GB</td>
<td>2,039 GB</td>
<td>$0.64/hr (~$476/yr)</td>
</tr>
</tbody>
</table>

**Graph:**

- **Y-axis:** Price (USD/Mon)
- **X-axis:** Standard A0 to D14
- **Legend:**
  - 100 Concurrent reqs.
What if I move the front-end from basic to standard tier?

- Expensive
- Slow
- High Effort

Cloud App + Service Configuration = 600 ms
What if I move the front-end from basic to standard tier?

Cloud App + Service Configuration = 200 ms

Cloud App + Service Configuration = 100 ms
Combining latency distributions
Combining latency distributions

WhenAll

task1  task2

Max
Combining latency distributions

WhenAny

task1  task2

Min
Cloud-Side Latency Estimation

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

Start

Frontend Processing

Store Insert

Blob Insert

Cache Insert

End

Replace from profile
WebPerf Approach

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;
}
Distributional Difference

WebPerf prediction

Ground truth from real deployment