Bridging the Gap between Serving and Analytics in Scalable Web Applications

Panagiotis Garefalakis
M.Res Thesis Presentation, 7 September 2015
Outline

• Motivation

• Challenges
  - Scalable web app design
  - Resource efficiency
  - Resource Isolation

• In-memory Web Objects model
  - Play2SDG case study
  - Experimental Results

• Conclusions

• Future work
Motivation

• Most modern web and mobile applications today offer highly **personalised** services generating **large** amounts of data.

• Tasks separated into **offline** (BE) and **online** (LC) based on the latency, computation and data freshness requirements.

• To train models and offer **analytics**, they use asynchronous offline computation, which leads to stale data being served to clients.

• To **serve** requests robustly and with low latency, applications cache data from the analytics layer.

• Applications deployed in **large clusters**, but with no collocation of tasks to avoid SLO violations.

• No data **freshness** guarantees and poor resource **efficiency**.
How does a typical scalable web application look like?

There is a strict decoupling of online and offline tasks.

With the emergence of cloud computing, these applications are deployed on clusters with thousands of machines.
Challenges: Resource Efficiency

- Most cloud facilities operate at very low utilisation, hurting both cost effectiveness and future scalability.

- Figure depicts a utilisation analysis for a production cluster at Twitter with thousands of servers, managed by Mesos over one month. The cluster mostly hosts user-centric services.

- The aggregate CPU utilisation is consistently below 20%, even though reservations reach up to 80% of total capacity.

Delimitrou, Christina, and Christos Kozyrakis. "Quasar: Resource-efficient and qos-aware cluster management. ASPLOS 2014"
Challenges: Resource Efficiency

- Even when looking at individual servers, their majority does not exceed 50% utilisation on any week.

- Typical memory use is higher (40-50%) but still differs from the reserved capacity.

Delimitrou, Christina, and Christos Kozyrakis. "Quasar: Resource-efficient and qos-aware cluster management. ASPLOS 2014"
Challenges: Resource Isolation

- Shared cluster environments suffer from resource interference. The main resources that are affected are CPU, caches (LLC), memory (DRAM), and network. There are also non-obvious interactions between resources, known as cross-resource interactions.

- What about resource isolation mechanisms provided by the Operating System - through scheduling?

- Even at low load, colocating LC with BE tasks creates sufficient pressure on the shared resources to lead to SLO violations. There are differences depending on the LC sensitivity on shared resources.

- The values are latencies, normalised to the SLO latency.

When a number of workloads execute concurrently on a server, they compete for shared resources.

In-memory Web Objects Model

- **IWOs**, express both online and offline logic of a web application as a single stateful distributed dataflow graph (SDG)
- **State** of the dataflow computation is expressed as IWOs, which are accessible as persistent objects by the application
- What about application strict **SLOs** - resource isolation and efficiency?

Tasks can be cooperatively scheduled, allowing to move resources between tasks of the dataflow efficiently according to the web application needs. As a result, the application can exploit data-parallel processing for compute-intensive requests and also maintain high resource utilisation, e.g. when training complex models, leading to fresher data while serving results with low latency from IWOs.
Play2SDG: Typical Web Music App

- Implemented a typical scalable web music service using Play Framework for Java
- **Decoupled** online and offline tasks to lower response latency
- **Asynchronous** collaborative filtering (CF) task using Apache Spark and Mesos for deployment

```scala
index(user, password){
  if(! User.authenticate(user, pass))
    return "Invalid credentials"
}
view(user){
  //Constructing Recommendation
  userRow = userItem.getRow(user)
  coOcc.multiply(userRow)
}
rate(user, item, rating){
  //Pushing new rating in the queue
  Queue.publish(user, item, rating)
}
```

- **Synchronous Task**
- **Asynchronous Task**
• Implemented a scalable web music service using IWOs API and making minor changes in the application code

• Express both online and offline logic of a web application as a stateful distributed dataflow graph

• Online collaborative filtering implementation using SDGs. addRating must achieve high throughput; getRec must serve requests with low latency, when recommendations are included in dynamically generated web pages

Front-End (Web)

```java
index(user, password){
    if(! User.authenticate(user, pass))
        return "Invalid credentials"
}
view(user){
    //Access Dataflow live state
    DataSource ds = DB.getDatasource()
    userRow = db.get(userItem).getRow(user)
    coOcc.multiply(userRow)
}
rate(user, item, rating){
    //Write directly to dataflow state
    DataSource ds = DB.getDatasource()
    ds.updateUserItem(user, item, rating)
    ds.updateCoOcc(userItem)
    return OK;
}
```

Back-End

SDG Distributed Processing System

Data Store

Cassandra

In-Memory Web Object (IWO)

Transparent State

userItem

CoOccurrence

Analytics Dataflow

SDG Distributed Processing System
Evaluation Platform

• Wombat’s private cluster with 5 machines

• Machines with 8 CPUs, 8 GB RAM and 1TB locally mounted disk, 1Gbps network

• Data: Million song Dataset, 943,347 unique tracks with 8,598,630 tag pairs

• **Software:**
  - Apache Spark 1.1.0
  - Apache Mesos is 0.22.1 (1 master node 3 slaves)
  - Nginx is 1.1.19
  - Cassandra database is 2.0.1

• **Load generator** is Apache JMeter 2.13 producing a specific functional behaviour pattern:
  1. user login
  2. navigate through the home page displaying the top 100 tracks
  3. visit the page with the latest recommendations
  4. user logout
Systems Compared

• **Isolated** Play2SDG
  
  • Play framework, Cassandra and Spark are configured to use up to 2 cores and 2GB of memory each through the Mesos API
  
  • Spark is set up in cluster mode and was **not allowed** to be colocated with Play application

• **Colocated** Play2SDG
  
  • Play framework, Cassandra and Spark are configured to use up to 2 cores and 2GB of memory each through the Mesos API
  
  • Spark is set up in cluster mode and was **allowed** to be colocated with Play application

• **Play2SDG IWOs** implementation
  
  • both serving and analytics tasks implemented as an SDG
  
  • configure the application JVM to use the same resources as above using JVM configuration and cgroups - disabled scheduling
Play2SDG Case Study Results

![Graph showing performance metrics for different configurations]

- **Isolated Play component with Cassandra**
- **Collocated Spark-Play components with Cassandra**
- **IWOs Serving and Analytics with Cassandra**

### Response time average (ms)

- **Isolated Play component with Cassandra**
- **Collocated Spark-Play components with Cassandra**
- **IWOs Serving and Analytics with Cassandra**

### Average Throughput (TPS)

- **Isolated Play component with Cassandra**
- **Collocated Spark-Play components with Cassandra**
- **IWOs Serving and Analytics with Cassandra**

### Response time percentiles (s)

- **Isolated 75th percentile**
- **Colocated 75th percentile**
- **IWOs 75th percentile**

- **Isolated 90th percentile**
- **Colocated 90th percentile**
- **IWOs 90th percentile**

- **Isolated 99th percentile**
- **Colocated 99th percentile**
- **IWOs 99th percentile**
Scheduling Results

Workload distribution across time

Response latency in ms

Scheduled serving IWOs

Time in seconds
Thesis Contributions

• Introduced **In-memory Web Objects** (IWOs), offering a unified model to developers when writing web applications that have the ability to serve data while using big data analytics.

• IWOs **isolation** mechanism that is based on cooperative task scheduling. Co-operative task scheduling reduces the scheduling decisions and allocates resources in a fine-grained way, leading to improved resource utilisation.

• The evaluation of IWOs by implementing **Play2SDG**, a real web application similar to Spotify, with both online/LC and offline/BE tasks. The web application was implemented as an extension of Play framework.
Future work

• Focus on efficient **distributed scheduling** of BE analytics and LC serving tasks

• Further investigate the **automatic conversion** of a web application in an SDG

• Implement IWOs abstract programming model for other stateful stream processing frameworks like Flink

• More Evaluation!
Questions???

Thank you!

email: pgaref@imperial.ac.uk
Demo time!
Backup Slide Isolation

Graph 1: Mesos deployment time and Application ramp up time vs. Number of instances

Graph 2: LXC container snapshot time and LXC container clone time vs. Number of instances
Backup Slide 2