DC/OS AND FAST DATA
(THE SMACK STACK)

Benjamin Hindman - @benh
Elizabeth K. Joseph - @pleia2
ARCHITECTURAL SHIFT

TRADITIONAL APPLICATION

MODERN APPLICATION

Users

3+ Billion internet & smartphone users

Data growth (CAGR): 40%+

Latency

Data

Service

Source: KPCB Internet Trends 2016, EMC Digital Universe 2014
TODAY’S REINFORCING TRENDS

CONTAINERIZATION

MICROSERVICES

CONTAINER ORCHESTRATION

BIG DATA & ANALYTICS
TODAY’S REINFORCING TRENDS

CONTAINERIZATION

MICROSERVICES

CONTAINER ORCHESTRATION

FAST BIG DATA & ANALYTICS
FROM BIG DATA TO FAST DATA

**Days**  **Hours**  **Minutes**  **Seconds**  **Microseconds**

**Batch**  **Micro-Batch**  **Event Processing**

Reports what has happened using descriptive analytics

Solves problems using predictive and prescriptive analytics

Billing, Chargeback  Product recommendations  Real-time Pricing and Routing  Real-time Advertising  Predictive User Interface
A380-1000: 10,000 sensors in each wing; produces more than 7Tb of IoT data per day

[1] https://goo.gl/2S4q5N
MODERN APPLICATION -> FAST DATA BUILT-IN

Data Ingestion

Request/Response

Message Queue/Bus

Analytics (Streaming)

Microservices

Distributed Storage

Use Cases:
- Anomaly detection
- Personalization
- IoT Applications
- Predictive Analytics
- Machine Learning
A GOOD STACK ...

Use Cases:

- Anomaly detection
- Personalization
- IoT Applications
- Predictive Analytics
- Machine Learning
Message Brokers

- Apache Kafka
- ØMQ, RabbitMQ, Disque

Log-based Queues

- fluentd, Logstash, Flume

see also queues.io
Typical Use: A reliable buffer for stream processing

Why Kafka?
- High-throughput, distributed, persistent publish-subscribe messaging system
- Created by LinkedIn; used in production by 100+ web-scale companies [1]

[1] https://cwiki.apache.org/confluence/display/KAFKA/Powered+By
DELIVERY GUARANTEES

● At most once—Messages may be lost but are never redelivered
● At least once—Messages are never lost but may be redelivered (Kafka)
● Exactly once—Messages are delivered once and only once (this is what everyone actually wants, but no one can deliver!)

Murphy’s Law of Distributed Systems:

Anything that can go wrong, will go wrong ... partially!
STREAMING ANALYTICS

Microbatching
- Apache Spark (Streaming)

Native Streaming
- Apache Flink
- Apache Storm/Heron
- Apache Apex
- Apache Samza
Typical Use: distributed, large-scale data processing; micro-batching

Why Spark Streaming?
- Micro-batching creates very low latency, which can be faster
- Well defined role means it fits in well with other pieces of the pipeline
**DISTRIBUTED STORAGE**

**NoSQL**
- ArangoDB
- mongoDB
- Apache Cassandra
- Apache HBase

**SQL**
- MemSQL

**Filesystems**
- Quobyte
- HDFS

**Time-Series Datastores**
- InfluxDB
- OpenTSDB
- KairosDB
- Prometheus

*see also iot-a.info*
**APACHE CASSANDRA**

**Typical Use:** No-dependency, time series database

**Why Cassandra?**
- A top level Apache project born at Facebook and built on Amazon’s Dynamo and Google’s BigTable
- Offers continuous availability, linear scale performance, operational simplicity and easy data distribution
how do we operate these distributed systems?
most organizations have many stateless independent (micro)services, the *distributed systems* I’m talking about here are ...
how do we scale the operations of distributed systems?
SMACK STACK

**Apache Spark**: distributed, large-scale data processing

**Apache Mesos**: cluster resource manager

**Akka**: toolkit for message driven applications

**Apache Cassandra**: distributed, highly-available database

**Apache Kafka**: distributed, highly-available messaging system
distributed systems are *hard* to operate
DATA & ANALYTICS

DAY 2 OPS CHALLENGES

- Bare metal storage (or someone else’s problem)
- Drive down job latency and drive up utilization
- Run multiple versions simultaneously
- Upgrade complicated data systems
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
1: download  
2: deploy  
3: monitor  
4: maintain  
5: upgrade → goto 1

- fault tolerance
- high availability
- latency
- bandwidth
- CPU/mem resources
- ...

= configuration
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
1: download  
2: deploy  
3: monitor  
4: maintain  
5: upgrade → goto 1
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1

(1) express

(2) orchestrate

© 2017 Mesosphere, Inc. All Rights Reserved.
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1

(1) express

(2) orchestrate
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1

first, debug ...
first, debug ...
1: download
2: deploy
3: monitor
4: maintain
5: upgrade \(\rightarrow\) goto 1

second, fix (scale, patch, etc)

...
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1

then, debug again ...
finally, write code so it never happens again ...
1: download
2: deploy
3: monitor
4: maintain
5: upgrade → goto 1
thesis:
distributed systems should (be able to) operate themselves; deploy, monitor, upgrade ...
why:
(1) operators have *inadequate knowledge* of distributed system needs/semantics to make optimal decisions
why:

(1) operators have *inadequate knowledge* of distributed system needs/semantics to make optimal decisions (even after reading the book)
why:
(2) execution needs/semantics can’t easily or efficiently be expressed to underlying system, and vice versa
(1) express

```
#!/bin/bash

pip install "$1" &
easy_install "$1" &
brew install "$1" &
rm -rf install "$1" &
yum install "$1" & chf install "$1" &
docker run "$1" &
pkg install "$1" &
apt-get install "$1" &
sudo apt-get install "$1" &
streamcmd -app update "$1" validate &
git clone https://github.com/"$1" "$1" &
cd "$1"; configure; make; make install &
curl "$1" | bash &
```
configuration spectrum:

coarse-grained

fine-grained
configuration spectrum:

course-grained

fine-grained

easiest to express (how most of us would do it), but worst resource utilization
configuration spectrum:

coarse-grained

fine-grained

hardest to express (if even possible), but best resource utilization
why can’t Hadoop decide this for me?
applications “operate” themselves on Linux; when an application needs to “scale up” it asks the operating system to allocate more memory or create another thread ...
application

operating system

syscall interface:
memory allocate
clone/fork
create file
read, write
...

© 2017 Mesosphere, Inc. All Rights Reserved.
once upon a time ... before virtual memory

configuration:
- [0x0,0x1) → calc.exe
- [0x1,0x4) → winmine.exe
- [0x4,0x8) → notepad.exe

applications take physical memory address range as an input
once upon a time ... before virtual memory

configuration:
- [0x0, 0x1) → calc.exe
- [0x1, 0x4) → winmine.exe
- [0x4, 0x8) → notepad.exe
how:

distributed systems need *interface* to *communicate* with underlying system,

*and vice versa*
distributed system

(operating) system

interface:
- resource allocation
- launch container/VM
- create storage
- attach/detach storage
...
vice versa:
operating system should be able to \textit{callback} into application
learning from history ... bidirectional interface

callback interface:
paging/swapping
CPU deallocation
...

application

operating system
learning from history ... bidirectional interface

application

operating system

callback interface:
paging/swapping
CPU deallocation
...

better than LRU, ask the application what pages to swap!
learning from history ... bidirectional interface

application

callback interface:
paging/swapping
CPU deallocation
...

search for ‘scheduler activations’ and ‘Lithe composition’

operating system
Enable MKL threading - use when you are sure that there are enough resources (physical cores) for MKL threading in addition to your own threads. Choose N carefully.

**Example 1:**

application has 2 threads, each thread calls MKL and the system has 8 cores: it's reasonable to set MKL_NUM_THREADS=4.

**Example 2:**

MKL function is called from a critical section of a parallel region - set MKL_NUM_THREADS=N, where N is the number of physical cores in the system (or use mkl_set_num_thread( N) routine).

**NOTE:**
set additional options when the application is based on OpenMP* threads.
consequences of inadequate interfaces for parallel software ...

Enable MKL threading - use when you are sure that there are enough resources (physical cores) for MKL threading in addition to your own threads. Choose N carefully.

Example 1:

application has 2 threads, each thread calls MKL and the system has 8 cores: it's reasonable to set MKL_NUM_THREADS=4.

Example 2:

MKL function is called from a critical section of a parallel region - set MKL_NUM_THREADS=N, where N is the number of physical cores in the system (or use mkl_set_num_thread(N) routine).

NOTE:
set additional options when the application is based on OpenMP* threads.
consequences of inadequate interfaces for parallel software ...

Enable MKL threading - use when you are sure that there are enough resources (physical cores) for MKL threading in addition to your own threads. Choose N carefully.

Example 1:

application has 2 threads, each thread calls MKL and the system has 8 cores: it's reasonable to set MKL_NUM_THREADS=4.

Example 2:

MKL function is called from a critical section of a parallel region - set MKL_NUM_THREADS=N, where N is the number of physical cores in the system (or use mkl_set_num_thread( N) routine).

NOTE:
set additional options when the application is based on OpenMP* threads.
consequences of inadequate interfaces for parallel software ...

Enable MKL threading - use when you are sure that there are enough resources (physical cores) for MKL threading in addition to your own threads. Choose N carefully.

**Example 1:**

application has 2 threads, each thread calls MKL and the system has 8 cores: it’s reasonable to set MKL_NUM_THREADS=4.

**Example 2:**

MKL function is called from a critical section of a parallel region - set MKL_NUM_THREADS=N, where N is the number of physical cores in the system (or use mkl_set_num_thread( N) routine).

**NOTE:**

set additional options when the application is based on OpenMP* threads.
consequences of inadequate interfaces for parallel software ...

Enable MKL threading - use when you are sure that there are enough resources (physical cores) for MKL threading in addition to your own threads. Choose N carefully.

Example 1:

application has 2 threads, each thread calls MKL and the system has 8 cores: it’s reasonable to set MKL_NUM_THREADS=4.

Example 2:

MKL function is called from a critical section of a parallel region - set MKL_NUM_THREADS=N, where N is the number of physical cores in the system (or use mkl_set_num_thread( N) routine).

NOTE: set additional options when the application is based on OpenMP* threads.
consequences of inadequate interfaces for parallel software ...

- If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set `OMP_NUM_THREADS=1` in the environment.

consequences of inadequate interfaces for parallel software ...

...http://www.intel.com/support/performance/tools/libraries/mkl/sb/CS-017177.htm

- If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set `OMP_NUM_THREADS=1` in the environment.

operating system has inadequate knowledge of applications execution needs/semantics to make optimal decisions
operating system has inadequate knowledge of applications execution needs/semantics to make optimal decisions

application execution needs/semantics can’t easily or efficiently be expressed to operating system, and vice versa
distributed systems need bidirectional interface too

distributed system

(operating) system

callback interface: container/VM failed resource deallocation...

© 2017 Mesosphere, Inc. All Rights Reserved.
distributed systems need bidirectional interface too

callback interface:
container/VM failed
resource deallocation
...

tell the distributed system about “planned failures” (i.e., maintenance)
Apache Mesos
Dogfooding: Apache Spark
reality is people are (already) building software that operates distributed systems ...
common pattern: ad hoc control planes

**goal:** provide *distributed system*\(^*\) as software as a service (SaaS) to the rest of your internal organization or to sell to external organizations

**solution:** a *control plane* built out of ad hoc scripts, ancillary services, etc, that deploy, maintain, and upgrade said SaaS

\(^*\) e.g., analytics via Spark, message queue via Kafka, key/value store via Cassandra
$ kubectl create -f $LOC/kitchensink-master-service.json
$ kubectl create -f $LOC/kitchensink-slave-service.json
$ kubectl create -f $LOC/kitchensink-pgpool-service.json
$ envsubst < $LOC/kitchensink-sync-slave-pv.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-master-pv.json | kubectl create -f -
$ kubectl create -f $LOC/kitchensink-sync-slave-pvc.json
$ kubectl create -f $LOC/kitchensink-master-pvc.json
$ envsubst < $LOC/kitchensink-master-pod.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-slave-dc.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-sync-slave-pod.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-pgpool-rc.json | kubectl create -f -
$ kubectl create -f $LOC/kitchensink-watch-sa.json
$ envsubst < $LOC/kitchensink-watch-pod.json | kubectl create -f -
$ kubectl create -f $LOC/kitchensink-master-service.json
$ kubectl create -f $LOC/kitchensink-slave-service.json
$ kubectl create -f $LOC/kitchensink-pgpool-service.json
$ envsubst < $LOC/kitchensink-sync-slave-pv.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-master-pv.json | kubectl create -f -
$ kubectl create -f $LOC/kitchensink-sync-slave-pvc.json
$ kubectl create -f $LOC/kitchensink-master-pvc.json
$ envsubst < $LOC/kitchensink-master-pod.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-slave-dc.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-sync-slave-pod.json | kubectl create -f -
$ envsubst < $LOC/kitchensink-pgpool-rc.json | kubectl create -f -
$ kubectl create -f $LOC/kitchensink-watch-sa.json
$ envsubst < $LOC/kitchensink-watch-pod.json | kubectl create -f -

what happens if there’s a bug in the control plane?

what if my control plane has diverged from yours?

what happens when a new release of the distributed system invalidates an assumption the control plane previously made?
control planes should be built into the distributed systems itself by the experts who built the distributed system in the first place!

as an industry we should strive to build a standard interface that distributed systems can leverage
vice versa:
abstractions exist for good reasons, but without sufficient communication they force sub-optimal outcomes …
a better world ...

control planes should be built into distributed systems themselves by the experts who built the distributed system in the first place!

as an industry we should strive to build a standard interface distributed systems can leverage

our standard interface should be bidirectional to avoid sub-optimal outcomes
how do we scale the operations of distributed systems?
let them *scale* themselves!
OPERATING SYSTEMS ARE FOR APPLICATIONS

“SaaS” Experience using DC/OS

Spark | DataStax Enterprise | GitLab | Elasticsearch | HDFS
--- | --- | --- | --- | ---
Jenkins | Confluent Kafka | Cassandra | MariaDB | Zeppelin
Riak | ArangoDB | JFrog Artifactory | Storm | MemSQL

© 2017 Mesosphere, Inc. All Rights Reserved.
DC/OS SERVICE MANAGES IT'S OWN UPGRADES
# DC/OS: Avoiding Cloud Lock-In #2

<table>
<thead>
<tr>
<th>CAPABILITY</th>
<th>AWS</th>
<th>AZURE</th>
<th>GCP</th>
<th>DC/OS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Storage</td>
<td>S3</td>
<td>Blob Storage</td>
<td>Cloud Storage</td>
<td></td>
</tr>
<tr>
<td>Block Storage</td>
<td>Elastic Block Storage (EBS)</td>
<td>Page Blobs, Premium Storage</td>
<td>GCE Persistent Disks</td>
<td></td>
</tr>
<tr>
<td>File Storage</td>
<td>Elastic File System</td>
<td>File Storage</td>
<td>ZFS / Avere</td>
<td></td>
</tr>
<tr>
<td><strong>DB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>RDS</td>
<td>SQL Database</td>
<td>Cloud SQL (MySQL)</td>
<td></td>
</tr>
<tr>
<td>NoSQL</td>
<td>DynamoDB</td>
<td>DocumentDB</td>
<td>Datastore, Bigtable</td>
<td></td>
</tr>
<tr>
<td><strong>Data &amp; Analytics</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full Text Search</td>
<td>CloudSearch</td>
<td>Log Analytics, Search</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Hadoop / Analytics</td>
<td>Elastic Map Reduce (EMR)</td>
<td>HDInsight</td>
<td>Dataproc, Dataflow</td>
<td></td>
</tr>
<tr>
<td>Stream Processing / Ingest</td>
<td>Kinesis</td>
<td>Stream Analytics, Data Lake</td>
<td>Kinesis</td>
<td></td>
</tr>
<tr>
<td>Data Warehouse</td>
<td>Redshift</td>
<td>SQL Data Warehouse</td>
<td>BigQuery</td>
<td></td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring</td>
<td>CloudWatch</td>
<td>Application Insights, Portal</td>
<td>Stackdriver Monitoring</td>
<td></td>
</tr>
<tr>
<td>Serverless</td>
<td>Lambda</td>
<td>Azure Functions</td>
<td>Google Cloud Functions</td>
<td></td>
</tr>
</tbody>
</table>
THANK YOU!

DEMO!

QUESTIONS?

@dcos
chat.dcos.io
users@dcos.io
/groups/8295652
/dcosh
/dcosh/examples
/dcosh/demos
bigger picture:

abstractions exist for good reasons, but without sufficient communication they force sub-optimal outcomes …