Lecture 8a
Introduction to BigData Analysis Stack (Hadoop 2.0)

Jianwen Wei
Center for HPC, SJTU
Apr 11th, 2018
Outline

• What is BigData?

• Hadoop Ecosystem

• Spark’s Improvement over Hadoop
What is Big Data?

“Big data are high volume, high velocity, and high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization” (Gartner 2012)

Volume
- exceeds limits of traditional column and row relational DB
- constantly growing

Requires Vertical scalability
- ability to grow storage to accommodate new ‘records’

Velocity
- arrives rapidly, often in real time

Requires Data streaming
- real time processing, analysis and transformation

Variety
- does not have a standard structure, e.g. text, images

Requires Horizontal scalability
- ability to add additional data structures
How is big data generated?

- Social media: posts, pictures and videos
- Sensors gathering information: e.g. Climate, traffic etc.
- Purchase transaction records
- High volume administrative & transactional records
- Digital satellite images
- Mobile phone GPS signals
Pilot 1: Smart meter project

Irish smart meter pilot study:
Single meter, total daily electricity consumption

Christmas 2009
Christmas 2010

Consecutive days with low consumption, possibly a week away?
Pilot 1: Smart meter project

Research Question: Investigate the potential of smart meter electricity data (high frequency – 30 mins) to identify household occupancy levels, potentially household structure

- England and Ireland both conducted pilots of rollout in 2009-2010 – data now available for research
- Southampton University commissioned by Beyond 2011 to conduct preliminary research (due mid Feb 2014)
Pilot 2: Mobile Phone Project

Manchester Catchment
Origin locations of all individuals present in Mcr City Centre – 17th April 2013
Pilot 2: Mobile Phone Project

Research Question: To investigate the possibility of using mobile phone data to model population flows, e.g., travel to work statistics

- Location data:
  - Telefonica proposal to provide aggregate data on origin-destination flows
- Requirement to engage with GDS before proceeding further
Pilot 3: Prices Project

Research Question: To investigate how we can scrape prices data from the internet and how this data could be used within price statistics

- 2-day workshop held with big data experts from Statistics Netherlands
- Focus on groceries
- Early prototype code in place
- Engagement with Billion Prices Project
Pilot 3: Prices by web scraping

Rendered webpage:

HTML code:

......
</div><div class="productLists" id="endFacets-1"><ul class="cf products line"><li id="p-254942348-3" class=" first"><div class="desc"><h3 class="inBasketInfoContainer"><a id="h-254942348" href="/groceries/Product/Details/?id=254942348" class="si_pl_254942348-title"><span class="image"><img src="http://img.tesco.com/Groceries/pi/121\5010044000121\IDShot_90x90.jpg" alt="" /></span> Warburtons Toastie Sliced White Bread 800G</a></h3><p class="limitedLife"><a href="http://www.tesco.com/groceries/zones/default.aspx?name=quality-and-freshness">Delivering the freshest food to your door- Find out more &gt;</a></p><div class="descContent"><div class="promo"><a href="/groceries/SpecialOffers/SpecialOfferDetail/Default.aspx?promoId=A31234788" title="All products available for this offer" id="flyout-254942348-promo-A31234788--pos" class="promoFlyout"> <span class="promoImgBox"><img src="/Groceries/UIAssets/I/Sites/Retail/Superstore/Online/Product/pos/2for.png" class="promoFlyout promo" alt="Special Offer" id="flyout-254942348-promo-A31234788--posimg" /></span><em>Any 2 for £2.00</em> <span>valid from 21/1/2014 until 10/2/2014</span></a></div><div class="tools"><div class="moreInfo"><a href="/groceries/Product/Details/?id=254942348" class="midiFlyout" id="flyout-254942348-midi-0-"><img class="midiFlyout hd" src="http://ui.tescoassets.com/groceries/UIAssets/I/../Compressed/I_635209615845382232/Sites/Retail/Superstore/Online/Product/infoBlue.gif" alt="" title="View product information" id="flyout-254942348-midi-1-" /></a></div><div class="links"><ul><li><a href="http://www.tesco.com/groceries/product/browse/default.aspx?notepad=white%20sliced%20loaf%20800g&N=4294793217" class="shelfFlyout active plaintooltip" id="s-tt-254942348" title="Premium White Bread"> Rest of <span class="hide">Premium White Bread !</span>shelf </a></li></ul></div></div></div><div class="quantity"><div class="content addToBasket"><p class="price"><span class="linePrice">£1.45<sup>!</sup></span><span class="linePriceAbbr"> (£0.18/100g)</span></p><h4 class="hide">Add to basket</h4><form method="post" id="fMultisearch-254942348".....

......

......
Pilot 3: The Billion Prices Project @ MIT

Lehman Brothers files for bankruptcy (15 Sept 2008)

Daily Online Price Index (United States)

Source: BPP - PriceStats - State Street
Pilot 4: Twitter Project

**Research Question:** To investigate how to capture geo-located tweets from Twitter and how this data might provide insights on commuting patterns and internal migration

- Opportunity to start experimenting early on with big data technologies
- Pilot work has successfully harvested geo-located tweets from the live Twitter feed using Python and Twitter API
- Need to determine whether planned application will exceed rate-limits
Pilot 4: Twitter Project

Temporal Patterns of International Mobility by selected country
Pilot 4: Mobility patterns from Twitter
Implication of BigData

- Analyzing the whole collection, not sampled ones.
- Long-tailed low-density heterogeneous data.
- Correlation, not causation.
Design Goals of MapReduce and Google File System

Building a *high-available, scalable, cost-economic and easy-to-use* programming environment on *unreliable* hardware.

The early stage of Google Data Center.
Hadoop is an OpenSource Implementation of MapReduce and GFS

Wordcount in MapReduce
HDFS Architecture

Metadata ops

Client

Datanodes

Rack 1

Rack 2

Replication

Block ops

Metadata (Name, #replicas, …): /users/foo/data, 3, …
Hadoop Ecosystem

- Pig
  - High-level language for data analysis
- HBase
  - Table storage for semi-structured data
- Hive
  - SQL-like Query language and Metastore
- Zookeeper
  - Coordinating distributed applications
- Mahout
  - Machine learning
Pig

- Started at Yahoo! Research
- Now runs about 30% of Yahoo!’s jobs
- Features
  - Expresses sequences of MapReduce jobs
  - Data model: nested “bags” of items
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc.)
  - Easy to plug in Java functions
An Example Problem

• Suppose you have user data in a file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25.

1. Load Users
2. Filter by age
3. Join on name
4. Group on url
5. Count clicks
6. Order by clicks
7. Take top 5
In MapReduce
In Pig Latin

Users = load ‘users’ as (name, age);
Filtered = filter Users by age >= 18 and age <= 25;
Pages = load ‘pages’ as (user, url);
Joined = join Filtered by name, Pages by user;
Grouped = group Joined by url;
Summed = foreach Grouped generate group,
        count(Joined) as clicks;
Sorted = order Summed by clicks desc;
Top5 = limit Sorted 5;
store Top5 into ‘top5sites’;
Ease of Translation

Job 1
- Load Users
  - Filter by age
    - Join on name

Job 2
- Load Pages
  - Users = load ...
  - Fltrd = filter ...
  - Pages = load ...
  - Joined = join ...
  - Grouped = group ...
  - Summed = ... count()...
  - Sorted = order ...
  - Top5 = limit ...

Job 3
- Group on url
- Count clicks
- Order by clicks
- Take top 5
Hive

- Developed at Facebook
- Used for majority of Facebook jobs
- “Relational database” built on Hadoop
  - Maintains list of **table schemas**
  - **SQL-like** query language (HiveQL)
  - Can call Hadoop Streaming scripts from HiveQL
  - Supports **table partitioning, clustering, complex data types, some optimizations**
Creating a Hive Table

CREATE TABLE page_views(viewTime INT, userid BIGINT,
                          page_url STRING, referrer_url STRING,
                          ip STRING COMMENT 'User IP address')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;

- Partitioning breaks table into separate files for each (dt, country) pair
  Ex: /hive/page_view/dt=2008-06-08,country=USA
      /hive/page_view/dt=2008-06-08,country=CA
A Simple Query

- Find all page views coming from xyz.com on March 31\textsuperscript{st}:

\[
\begin{align*}
\text{SELECT} & \quad \text{page\_views.}\ast \\
\text{FROM} & \quad \text{page\_views} \\
\text{WHERE} & \quad \text{page\_views.date} \geq '2008-03-01' \\
\text{AND} & \quad \text{page\_views.date} \leq '2008-03-31' \\
\text{AND} & \quad \text{page\_views.referrer\_url} \text{ like } '%xyz.com';
\end{align*}
\]

- Hive only reads partition 2008-03-01,* instead of scanning entire table
Aggregation and Joins

• Count users who visited each page by gender:

```sql
SELECT pv.page_url, u.gender, COUNT(DISTINCT u.id)
FROM page_views pv JOIN user u ON (pv.userid = u.id)
GROUP BY pv.page_url, u.gender
WHERE pv.date = '2008-03-03';
```

• Sample output:

<table>
<thead>
<tr>
<th>page_url</th>
<th>gender</th>
<th>count(userid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>home.php</td>
<td>MALE</td>
<td>12,141,412</td>
</tr>
<tr>
<td>home.php</td>
<td>FEMALE</td>
<td>15,431,579</td>
</tr>
<tr>
<td>photo.php</td>
<td>MALE</td>
<td>23,941,451</td>
</tr>
<tr>
<td>photo.php</td>
<td>FEMALE</td>
<td>21,231,314</td>
</tr>
</tbody>
</table>
Using a Hadoop *Streaming* Mapper Script

```sql
SELECT TRANSFORM(page_views.userid, page_views.date)
USING 'map_script.py'
AS dt, uid CLUSTER BY dt
FROM page_views;
```
HBase - What?

- Modeled on Google’s Bigtable
- Row/column store
- Billions of rows/millions on columns
- Column-oriented - nulls are free
- Untyped - stores byte[]
## HBase - Data Model

<table>
<thead>
<tr>
<th>Row</th>
<th>Timestamp</th>
<th>Column family: animal</th>
<th>Column family repairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>enclosure1</td>
<td>t2</td>
<td>zebra</td>
<td>1000 EUR</td>
</tr>
<tr>
<td></td>
<td>t1</td>
<td>lion</td>
<td>big</td>
</tr>
<tr>
<td>enclosure2</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
HBase - Data Storage

Column family animal:

<table>
<thead>
<tr>
<th>Enclosure</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>enclosure1</td>
<td>zebra</td>
</tr>
<tr>
<td>enclosure1</td>
<td>big</td>
</tr>
<tr>
<td>enclosure1</td>
<td>lion</td>
</tr>
</tbody>
</table>

Column family repairs:

<table>
<thead>
<tr>
<th>Enclosure</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>enclosure1</td>
<td>1000 EUR</td>
</tr>
</tbody>
</table>
HTable table = ...
Text row = new Text("enclosure1");
Text col1 = new Text("animal:type");
Text col2 = new Text("animal:size");
BatchUpdate update = new BatchUpdate(row);
update.put(col1, "lion".getBytes("UTF-8"));
update.put(col2, "big".getBytes("UTF-8"));
table.commit(update);

update = new BatchUpdate(row);
update.put(col1, "zebra".getBytes("UTF-8"));
table.commit(update);
HBase - Querying

• Retrieve a cell
  Cell = table.getRow("enclosure1").getColumn("animal:type").getValue();

• Retrieve a row
  RowResult = table.getRow("enclosure1");

• Scan through a range of rows
  Scanner s = table.getScanner(new String[]{"animal:type"});
Limitations of Hadoop

- Needs *synchronization before shuffle*, in which the slowest task may become the bottleneck.

- Needs *serialization between iteration*, which is unfit for iterative jobs.

- **Batch** jobs is NOT optimized for streaming interactive queries.

- Do NOT utilize *memory* efficiently.
Spark

In-Memory Cluster Computing for Iterative and Interactive Applications

Matei Zaharia, Mosharaf Chowdhury, Justin Ma, Michael Franklin, Scott Shenker, Ion Stoica
Motivation

• Acyclic data flow is a powerful abstraction, but is not efficient for applications that repeatedly reuse a *working set* of data:
  – *Iterative* algorithms (many in machine learning)
  – *Interactive* data mining tools (R, Excel, Python)

• Spark makes working sets a first-class concept to efficiently support these apps
Spark Goal

• Provide distributed memory abstractions for clusters to support apps with working sets

• Retain the attractive properties of MapReduce:
  – Fault tolerance (for crashes & stragglers)
  – Data locality
  – Scalability

Solution: augment data flow model with “resilient distributed datasets” (RDDs)
Generality of RDDs

• We conjecture that Spark’s combination of data flow with RDDs unifies many proposed cluster programming models
  – *General data flow models*: MapReduce, Dryad, SQL
  – *Specialized models for stateful apps*: Pregel (BSP), HaLoop (iterative MR), Continuous Bulk Processing

• Instead of specialized APIs for one type of app, give user first-class control of distrib. datasets
Next-gen Big Data Processing Goals

• **Low latency (interactive) queries on historical data**: enable faster decisions
  – E.g., identify why a site is slow and fix it

• **Low latency queries on live data (streaming)**: enable decisions on real-time data
  – E.g., detect & block worms in real-time (a worm may infect 1mil hosts in 1.3sec)

• **Sophisticated data processing**: enable “better” decisions
  – E.g., anomaly detection, trend analysis
Outline

• Spark programming model
• Example applications
• Implementation
• Demo
• Future work
Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be cached across parallel operations

- Parallel operations on RDDs
  - Reduce, collect, count, save, …

- Restricted shared variables
  - Accumulators, broadcast variables
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```scala
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split("\t")(2))
cachedMsgs = messages.cache()
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
...
```

Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
RDDs in More Detail

• An RDD is an immutable, partitioned, logical collection of records
  » Need not be materialized, but rather contains information to rebuild a dataset from stable storage

• Partitioning can be based on a key in each record (using hash or range partitioning)

• Built using bulk transformations on other RDDs

• Can be cached for future reuse
# RDD Operations

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Parallel operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(define a new RDD)</td>
<td>(return a result to driver)</td>
</tr>
<tr>
<td>map</td>
<td>reduce</td>
</tr>
<tr>
<td>filter</td>
<td>collect</td>
</tr>
<tr>
<td>sample</td>
<td>count</td>
</tr>
<tr>
<td>union</td>
<td>save</td>
</tr>
<tr>
<td>groupByKey</td>
<td>lookupKey</td>
</tr>
<tr>
<td>reduceByKey</td>
<td>...</td>
</tr>
<tr>
<td>join</td>
<td></td>
</tr>
<tr>
<td>cache</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
RDD Fault Tolerance

- RDDs maintain *lineage* information that can be used to reconstruct lost partitions

**Ex:** cachedMsgs = textFile(...).filter(_.contains("error")).map(_.split('\t')(2)).cache()
Benefits of RDD Model

• Consistency is easy due to immutability
• Inexpensive fault tolerance (log lineage rather than replicating/checkpointing data)
• Locality-aware scheduling of tasks on partitions
• Despite being restricted, model seems applicable to a broad variety of applications
### RDDs vs Distributed Shared Memory

<table>
<thead>
<tr>
<th>Concern</th>
<th>RDDs</th>
<th>Distr. Shared Mem.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reads</td>
<td>Fine-grained</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Writes</td>
<td>Bulk transformations</td>
<td>Fine-grained</td>
</tr>
<tr>
<td>Consistency</td>
<td>Trivial (immutable)</td>
<td>Up to app / runtime</td>
</tr>
<tr>
<td>Fault recovery</td>
<td>Fine-grained and low-overhead using lineage</td>
<td>Requires checkpoints and program rollback</td>
</tr>
<tr>
<td>Straggler mitigation</td>
<td>Possible using speculative execution</td>
<td>Difficult</td>
</tr>
<tr>
<td>Work placement</td>
<td>Automatic based on data locality</td>
<td>Up to app (but runtime aims for transparency)</td>
</tr>
</tbody>
</table>
Related Work

- **DryadLINQ**
  » Language-integrated API with SQL-like operations on lazy datasets
  » Cannot have a dataset persist across queries
- **Relational databases**
  » Lineage/provenance, logical logging, materialized views
- **Piccolo**
  » Parallel programs with shared distributed tables; similar to distributed shared memory
- **Iterative MapReduce (Twister and HaLoop)**
  » Cannot define multiple distributed datasets, run different map/reduce pairs on them, or query data interactively
- **RAMCloud**
  » Allows random read/write to all cells, requiring logging much like distributed shared memory systems
Outline

• Spark programming model
• Example applications
• Implementation
• Demo
• Future work
Example: Logistic Regression

• Goal: find best line separating two sets of points
Logistic Regression Code

- val data = spark.textFile(...).map(readPoint).cache()

- var w = Vector.random(D)

- for (i <- 1 to ITERATIONS) {
  - val gradient = data.map(p => (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x)
  - .reduce(_ + _)
  - w += gradient
- }

- println("Final w: " + w)
Logistic Regression
Performance

- First iteration: 174 s
- Further iterations: 6 s
- Average iteration: 127 s

![Bar chart showing running time for Hadoop and Spark across iterations]

- Hadoop
- Spark

Number of Iterations vs. Running Time (s)
Example: MapReduce

- MapReduce data flow can be expressed using RDD transformations

```scala
res = data.flatMap(rec => myMapFunc(rec))
  .groupByKey()
  .map((key, vals) => myReduceFunc(key, vals))
```

Or with combiners:

```scala
res = data.flatMap(rec => myMapFunc(rec))
  .reduceByKey(myCombiner)
  .map((key, val) => myReduceFunc(key, val))
```
Word Count in Spark

```scala
val lines = spark.textFile("hdfs://...")

val counts = lines.flatMap(_.split("\s"))
  .reduceByKey(_ + _)

counts.save("hdfs://...")
```
Example: Pregel

- Graph processing framework from Google that implements Bulk Synchronous Parallel model
- Vertices in the graph have state
- At each superstep, each node can update its state and send messages to nodes in future step
- Good fit for PageRank, shortest paths, …
Pregel Data Flow

Input graph ➔ Vertex state 1 ➔ Messages 1 ➔ Superstep 1 ➔ Group by vertex ID

Input graph ➔ Vertex state 2 ➔ Messages 2 ➔ Superstep 2 ➔ Group by vertex ID

...
PageRank in Pregel

Input graph → Vertex ranks 1 → Superstep 1 (add contribs) → Vertex ranks 2 → Superstep 2 (add contribs) → ...

Contributions:
1. Group & add by vertex
2. Group & add by vertex
Pregel in Spark

• Separate RDDs for immutable graph state and for vertex states and messages at each iteration
• Use groupByKey to perform each step
• Cache the resulting vertex and message RDDs
• Optimization: co-partition input graph and vertex state RDDs to reduce communication
Other Spark Applications

- Twitter spam classification (Justin Ma)
- EM alg. for traffic prediction (Mobile Millennium)
- K-means clustering
- Alternating Least Squares matrix factorization
- In-memory OLAP aggregation on Hive data
- SQL on Spark (future work)
Language Integration

• Scala closures are Serializable Java objects
  » Serialize on driver, load & run on workers
• Not quite enough
  » Nested closures may reference entire outer scope
  » May pull in non-Serializable variables not used inside
  » Solution: bytecode analysis + reflection
• Shared variables implemented using custom serialized form (e.g. broadcast variable contains pointer to BitTorrent tracker)
Interactive Spark

• Modified Scala interpreter to allow Spark to be used interactively from the command line

• Required two changes:
  – Modified wrapper code generation so that each “line” typed has references to objects for its dependencies
  – Place generated classes in distributed filesystem

• Enables in-memory exploration of big data
Conclusion

• By making distributed datasets a first-class primitive, Spark provides a simple, efficient programming model for stateful data analytics

• RDDs provide:
  » Lineage info for fault recovery and debugging
  » Adjustable in-memory caching
  » Locality-aware parallel operations

• We plan to make Spark the basis of a suite of batch and interactive data analysis tools
Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

\[
\begin{align*}
\text{lines} &= \text{spark.textFile("hdfs://...")} \\
\text{errors} &= \text{lines.filter(_.startsWith("ERROR"))} \\
\text{messages} &= \text{errors.map(_.split(\'\t\')(2))} \\
\text{cachedMsgs} &= \text{messages.cache()} \\
\text{cachedMsgs.filter(_.contains("foo")).count} \\
\text{cachedMsgs.filter(_.contains("bar")).count} \\
\end{align*}
\]

**Result:** full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)
Architecture of Big Data Analytics

Big Data Sources
- * Internal
- * External
- * Multiple formats
- * Multiple locations
- * Multiple applications

Big Data Transformation
- Middleware
- Extract Transform Load
- Data Warehouse
- Traditional Format CSV, Tables

Big Data Platforms & Tools
- Hadoop
- MapReduce
- Pig
- Hive
- Jaql
- Zookeeper
- Hbase
- Cassandra
- Oozie
- Avro
- Mahout
- Others

Big Data Analytics Applications
- Queries
- Reports
- OLAP
- Data Mining

Source: Stephan Kudyba (2014), Big Data, Mining, and Analytics: Components of Strategic Decision Making, Auerbach Publications
Summary

• Support **interactive** and **streaming** computations
  – In-memory, fault-tolerant storage abstraction, low-latency scheduling,...
• **Easy** to combine **batch**, **streaming**, and **interactive** computations
  – Spark execution engine supports all comp. models
• **Easy** to develop **sophisticated** algorithms
  – Scala interface, APIs for Java, Python, Hive QL, ...
  – New frameworks targeted to graph based and ML algorithms
• **Compatible** with existing open source ecosystem
• **Open source** (Apache/BSD) and fully committed to release **high quality** software
  – Three-person software engineering team lead by Matt Massie (creator of Ganglia, 5th Cloudera engineer)
### BigData v.s. HPC

<table>
<thead>
<tr>
<th></th>
<th>HPC</th>
<th>BigData (Before Spark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>RAID SAN &lt;100PB Cap &gt;100GB BW POSIX-compatible</td>
<td>Distributed Storage with data replication &gt;100PB Cap &lt;100GB BW Non POSIX-compatible</td>
</tr>
<tr>
<td>Network</td>
<td>RDMA Infiniband</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Programming Model</td>
<td>MPI (tightly coupled)</td>
<td>MapReduce (loosely coupled)</td>
</tr>
<tr>
<td>Language</td>
<td>C, FORTRAN</td>
<td>Java, Python</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>Checkpoint</td>
<td>Recomputation between tasks</td>
</tr>
<tr>
<td>Bottleneck</td>
<td>Computation or Mem-BW bound</td>
<td>I/O bound</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Schedule tasks to CPUs in the same closet</td>
<td>Schedule tasks to data storage</td>
</tr>
</tbody>
</table>