Apache Sqoop: Highlights of Sqoop 2

Kathleen Ting (kate at cloudera dot com)
Customer Operations Engineer

January 2012

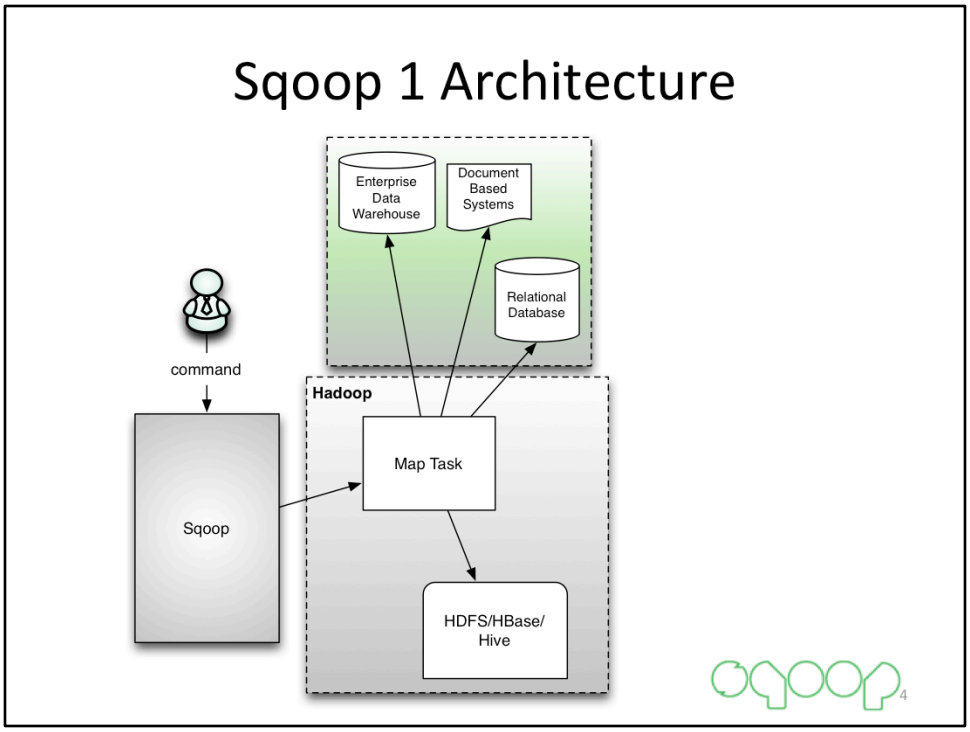
Because databases are not easily accessible by Hadoop, Apache Sqoop was created to efficiently transfer bulk data between Hadoop and external structured datastores. The popularity of Sqoop in enterprise systems confirms that Sqoop does bulk transfer admirably. That said, to enhance its functionality, Sqoop needs to fulfill data integration use-cases as well as become easier to manage and operate.

Sqoop is currently undergoing incubation at The Apache Software Foundation.

More information on this project can be found at http://incubator.apache.org/sqoop.

What is Sqoop?

- Bulk data transfer tool
  - Import/Export from relational database, enterprise data warehouse, NoSQL systems
  - Populate tables in Hive, HBase
  - Schedule Oozie automated import/export tasks
  - Support plugins via Connector based architecture



The dataset being transferred is sliced up into different partitions and a map-only job is launched with individual mappers responsible for transferring a slice of this dataset. Each record of the data is handled in a type safe manner since Sqoop uses the database metadata to infer the data types.

Sqoop 1 launches a single Map-only Job that does both data transport and transform. A MR job imports a table from a db, extracts rows from the table, and writes the records to HDFS. Sqoop then integrates into Hive/HBase, or goes through format conversions, compression, partitioning, indexing.

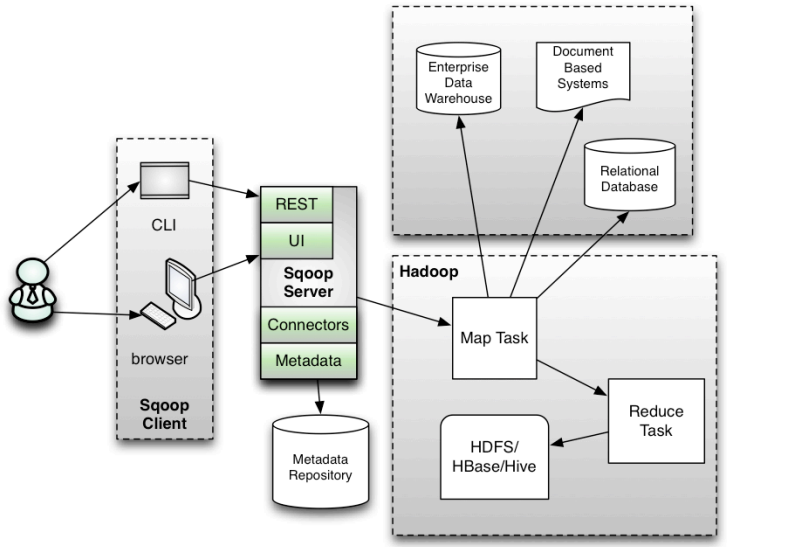
SQL to Hadoop Tool
- Import/Export from relational db, enterprise data warehouse, NoSQL systems
- Populate tables in Hive, HBase
- Support plugins via Connector based architecture
- Sqoop imports tables from db into HDFS for deep analysis
- Sqoop exports MR results back to a db for presentation to end-users
- Sqoop can import/export in HDFS; Sqoop can only import into Hive, HBase

Sqoop 1 Challenges

- Cryptic, contextual command line arguments
- Tight coupling between data transfer and serialization format
- Security concerns with openly shared credentials
- Not easy to manage config/install
- Not easy to monitor map job
- Connectors are forced to follow JDBC model

- Cryptic e.g. Error-prone connector matching, and since it is not enforced, can cause user errors
- Tight coupling e.g. direct MySQL connector can't support sequence files
- Not easy to manage e.g. local configuration requires root privileges
- Not easy to monitor e.g. verbose flag
- Different connectors interpret these options differently. Some options are not understood for the same operation by different connectors, while some connectors have custom options that do not apply to others. Confusing for users and detrimental to effective use.
- Some connectors may support a certain data format while others don't – connector should only focus on connectivity and serialization, format conversion, Hive/HBase integration should be uniformly available via Sqoop framework.
- Required to use common JDBC vocabulary (URL, database, table, etc.)

Sqoop 2 Architecture



6

Agenda

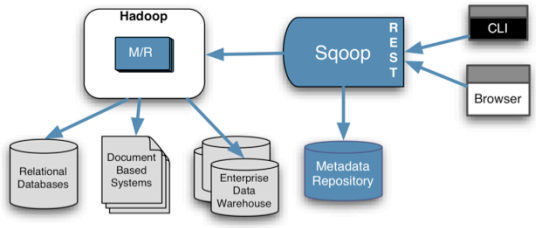
- Ease of Use
  - Sqoop 1: Client-side Tool
  - Sqoop 2: Sqoop as a Service
  - Client Interface
  - Sqoop 1: Service Level Integration
  - Sqoop 2: Service Level Integration
- Ease of Extension
  - Sqoop 1: Implementing Connectors
  - Sqoop 2: Implementing Connectors
  - Sqoop 1: Using Connectors
  - Sqoop 2: Using Connectors
- Security
  - Sqoop 1: Security
  - Sqoop 2: Security
  - Sqoop 1: Accessing External Systems
  - Sqoop 2: Accessing External Systems
  - Sqoop 1: Resource Management
  - Sqoop 2: Resource Management

Sqoop 1: Client-side Tool

- Sqoop 1 is a client-side tool
  - Client-side installation + configuration
    - Connectors locally installed
    - Local configuration, requiring root privileges
    - JDBC drivers needed locally
    - Database connectivity needed locally

Sqoop 2: Sqoop as a Service

- Server-side installation + configuration
  - Connectors configured in one place, managed by Admin/run by Operator
  - JDBC drivers in one place
  - Database connectivity needed on the server



9

Sqoop as a web-based service, exposes the REST API
- Front-ended by CLI and browser
- Back-ended by a metadata repository

Example of document based system is couchbase.

Sqoop 1 has something called a sqoop metastore, which is similar to a repository for metadata but not quite. That said, the model of operation for Sqoop 1 and Sqoop 2 is very different: Sqoop 1 was a limited vocabulary tool while Sqoop 2 is more metadata driven. The design of Sqoop 2's metadata repository is such that it can be replaced by other providers.

Client Interface

- Sqoop 1 client interface:
  - Command-Line Interface (CLI) based, thus scriptable
- Sqoop 2 client interface:
  - CLI based, thus scriptable
  - Web based, thus accessible
  - REST API exposed for external tool integration

Not bound by terminal, well documented return codes

Sqoop 1: Service Level Integration

- Hive, HBase
  - Requires local installation
- Oozie
  - von Neumann(esque) integration:
    - Packaged Sqoop as an action
    - Then ran Sqoop from node machines, causing one MR job to be dependent on another MR job
    - Error-prone, difficult to debug

Oozie launches Sqoop by bundling it and running it on the cluster

Sqoop 2: Service Level Integration

- Hive, HBase
  - Server-side integration
- Oozie
  - REST API integration

- Hive, HBase – integration happens not from client but from the backend – connectivity required in backend; put behind dmz
- Decouple oozie and sqoop, if install new sqoop connector then don't need to install it in oozie also
- Hive will not invoke anything in Sqoop, while Oozie does invoke Sqoop so the REST API does not benefit Hive in any way but it does benefit Oozie
- Hive does not need to be installed on Sqoop at all. What Sqoop will do is submit requests to the HiveServer over the wire
- Which Hive/HBase server the data will be put into is the responsibility of the reduce phase which will have its own configuration and since both these systems have are on Hadoop - we don't need any added security besides passing down the Kerberos principal

Ease of Use (summary)

Sqoop 1	Sqoop 2
Client-side install	Server-side install
CLI based	CLI + Web based
Client access to Hive, HBase	Server access to Hive, HBase
Oozie and Sqoop tightly coupled	Oozie finds REST API

Agenda

- Ease of Use
  - Sqoop 1: Client-side Tool
  - Sqoop 2: Sqoop as a Service
  - Client Interface
  - Sqoop 1: Service Level Integration
  - Sqoop 2: Service Level Integration
- Ease of Extension
  - Sqoop 1: Implementing Connectors
  - Sqoop 2: Implementing Connectors
  - Sqoop 1: Using Connectors
  - Sqoop 2: Using Connectors
- Security
  - Sqoop 1: Security
  - Sqoop 2: Security
  - Sqoop 1: Accessing External Systems
  - Sqoop 2: Accessing External Systems
  - Sqoop 1: Resource Management
  - Sqoop 2: Resource Management

Sqoop 1: Implementing Connectors

- Connectors forced to follow JDBC model
  - Connectors limited/required to use common JDBC vocabulary (URL, database, table, etc)
- Connectors must implement all Sqoop functionality that they want to support
  - New functionality not avail for old connectors

- Not easy to work with non relational db
- Heavily JDBC centric
- Couchbase implementation required different interpretation
- Inconsistencies between connectors

Sqoop 2: Implementing Connectors

- Connectors are not restricted to JDBC model
  - Connectors can define own vocabulary
- Common functionality abstracted out of connectors
  - Connectors only responsible for data transport
  - Common Reduce phase implements functionality
  - Ensures that connectors benefit from future dev of functionality

Two-phases: first, transfer; second, transform/integration with other components
Option to opt-out of downstream processing (i.e. revert to Sqoop 1)
Trade-off between ease of connector/tooling development vs faster performance
Separating data transfer (Map) from data transform (Reduce) allows connectors to specialize
Connectors benefit from a common framework of functionality
Functionally, Sqoop 2 is a superset of Sqoop 1 but does it in a different way
Too early in the design process to tell if the same CLI commands could be used but most likely not primarily because it is a fundamentally incompatible change
Reduce phase limited to stream transformations (no aggregation to start with)

Different Options, Different Results

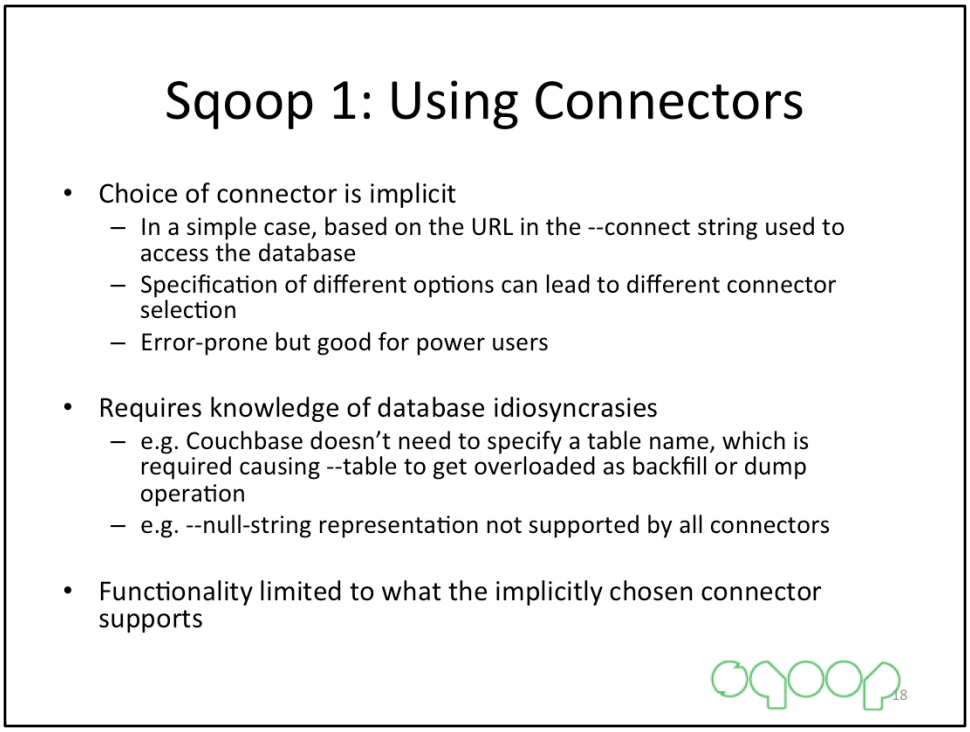
Which is running MySQL?

```
$ sqoop import --connect jdbc:mysql://localhost/db \
--username foo --table TEST

$ sqoop import --connect jdbc:mysql://localhost/db \
--driver com.mysql.jdbc.Driver --username foo --table TEST
```

- Different options can lead to unpredictable results
  - Sqoop 2 requires explicit selection of connector thus disambiguating the process

Former is running MySQL b/c specifying driver option prevents the MySQL connector from working i.e. would end up using generic JDBC connector



Based on the URL in the connect string used to access the database, Sqoop attempts to predict which driver it should load.

What are connectors?

- Plugin components based on Sqoop's extension framework
- Efficiently transfer data between Hadoop and external store
- Meant for optimized import/export or don't support native JDBC
- Bundled connectors: MySQL, PostgreSQL, Oracle, SQLServer, JDBC
- High-performance data transfer: Direct MySQL, Direct PostgreSQL

Sqoop 2: Using Connectors

- User makes explicit connector choice
  - Less error-prone, more predictable
- User need not be aware of the functionality of all connectors
  - Couchbase users need not care that other connectors use tables
- Common functionality available to all connectors
  - Connectors need not worry about downstream functionality, transformations, integration with other systems

Add an interactive UI
- Walk-through import/export setup, which eliminates redundant/incorrect options
- Various connectors are added in one place; connectors expose necessary options to Sqoop framework
- User only required to provide info relevant to their use-case

Ease of Extension (summary)

Sqoop 1	Sqoop 2
Connector forced to follow JDBC model	Connector given free rein
Connectors must implement functionality	Connectors benefit from common framework of functionality
Connector selection is implicit	Connector selection is explicit

Agenda

- Ease of Use
  - Sqoop 1: Client-side Tool
  - Sqoop 2: Sqoop as a Service
  - Client Interface
  - Sqoop 1: Service Level Integration
  - Sqoop 2: Service Level Integration
- Ease of Extension
  - Sqoop 1: Implementing Connectors
  - Sqoop 2: Implementing Connectors
  - Sqoop 1: Using Connectors
  - Sqoop 2: Using Connectors
- Security
  - Sqoop 1: Security
  - Sqoop 2: Security
  - Sqoop 1: Accessing External Systems
  - Sqoop 2: Accessing External Systems
  - Sqoop 1: Resource Management
  - Sqoop 2: Resource Management

Sqoop 1: Security

- Inherits/propagates Kerberos principal for the jobs it launches
- Access to files on HDFS can be controlled via HDFS security
- Sqoop operates as command line Hadoop client
- No support for securing access to external systems
  - E.g. relational database

Sqoop 2: Security

- Inherits/propagates Kerberos principal for the jobs it launches
- Access to files on HDFS can be controlled via HDFS security
- Sqoop operates as server based application
- Support for securing access to external systems via role-based access to Connection objects
  - Admins create/edit/delete Connections
  - Operators use Connections
- Audit trail logging

No code generation, no compilation allows Sqoop to run where there are no compilers, which makes it more secure by preventing bad code from running
Previously required direct access to Hive/HBase
More secure because routed through Sqoop server rather than opening up access to all clients to perform jobs

Sqoop 1: Accessing External Systems

- Every invocation requires necessary credentials to access external systems (e.g. relational database)
  - Workaround: Admin creates a limited access user in lieu of giving out password
    - Doesn't scale
    - Permission granularity is hard to obtain
- Hard to prevent misuse once credentials are given

Sqoop 2: Accessing External Systems

- Sqoop 2 introduces Connections as First-Class Objects
  - Connection encompass credentials
  - Connections created once, then used many times for various import/export Jobs
  - Connections created by Admin, used by Operator
    - Safeguard credential access from end user
- Restrict scope: connections can be restricted based on operation (import/export)
  - Operators cannot abuse credentials

Connection is only for external systems

Sqoop 1: Resource Management

- No explicit resource management policy
  - User specifies number of map jobs to run
  - Can't throttle load on external systems

Sqoop 2: Resource Management

- Connections allow specification of resource policy
  - Admin can limit the total number of physical Connections open at one time
  - Connections can be disabled

No need to disable user in database

Security (summary)

Sqoop 1	Sqoop 2
Support only for Hadoop security	Support for Hadoop security and role-based access control to external systems
High risk of abusing access to external systems	Reduced risk of abusing access to external systems
No resource management policy	Resource management policy

Takeaway

Sqoop 2 Highights:

- Ease of Use: Sqoop as a Service
- Ease of Extension: Connectors benefit from shared functionality
- Security: Connections as First-Class objects, Role-based Security

Current Status: work-in-progress

- Sqoop 2 Development: https://issues.apache.org/jira/browse/SQOOP-365
- Sqoop 2 Design: https://cwiki.apache.org/confluence/display/SQOOP/Sqoop+2