Tutorial: Sector/Sphere Installation and Usage

Yunhong Gu

July 2010

Agenda

- System Overview
- Installation
- File System Interface
- Sphere Programming
- Conclusion

The Sector/Sphere Software

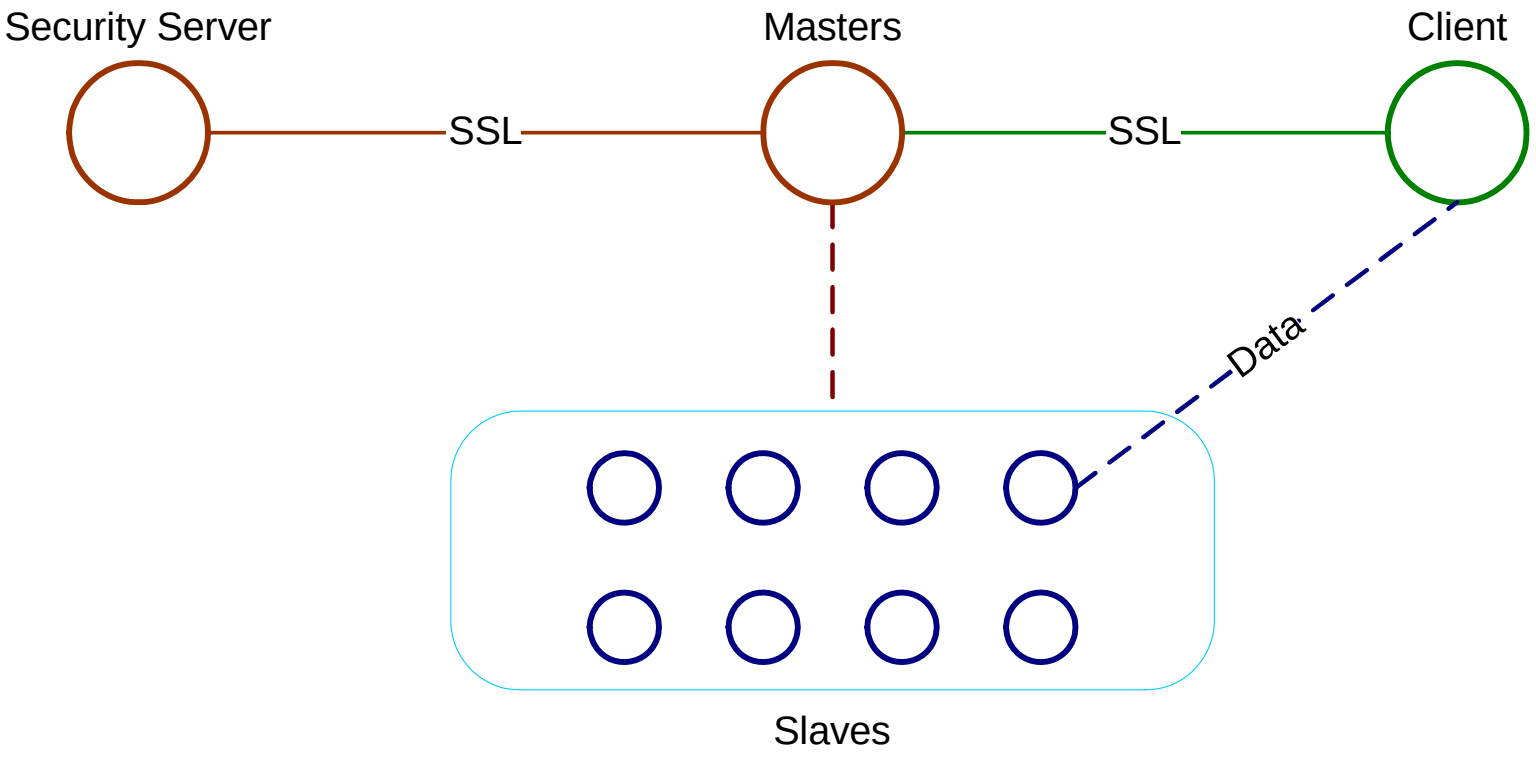
- Open Source, BSD/Apache license, available from http://sector.sf.net
- Developed in C++
- Includes two components:
  - Sector distributed file system
  - Sphere parallel data processing framework
- Current version is 2.4

Why Sector/Sphere

- Sector distributed file system
  - High performance, scalable user space file system running on cluster of commodity computers
  - Support wide area networks
  - Application-aware
  - Compatible with legacy systems
  - Content distribution/collection/sharing

- Sphere parallel data processing framework
  - Massive parallel in-storage processing based data locality
  - Simplified API with UDF applied to data segments in parallel
  - Transparent load balancing and fault tolerance
  - Faster than Hadoop MapReduce by 2 – 4x

System Overview



System Components

- Security server
  - Maintain user accounts and other security policies, such as IP ACL
  - Sector uses its own user accounts, but will be expandable to connect to other security systems
- Master server
  - Maintain metadata and manage file system running, accepts users' requests
  - Multiple master servers can be started for load balancing and high availability

System Components (cont.)

- Slave
  - Commodity computers with internal disks and Gb/s or 10Gb/s network connections
  - Sector uses Slave's native file system (e.g., ext3, xfs, etc.) to store data
- Client
  - Includes libraries, header files, and tools to access the Sector system and develop applications

System Requirements

- Sector server side works on Linux only
  - Windows servers will be available in version 2.5 or 2.6
- Sector client works on Linux and Windows
- On Linux, the system requires g++ version 3.4 or above and openssl development library (libssl-dev or openssl-devel)
- In this tutorial we will only explain the installation on Linux

Code Structure

- conf : configuration files
- tools: client tools
- doc: Sector documentation
- include: programming header files (C++)
- security: security server
- Makefile
- examples: Sphere programming examples
- lib: places to stored compiled libraries
- slave: slave server
- fuse: FUSE interface
- master: master server

Installation

- Documentation: http://sector.sourceforge.net/doc/index.htm

- Download sector.2.4.tar.gz from Sector SourceForge project website
- tar –zxvf sector.2.4.tar.gz
- ./sector-sphere
  - run “make”
- RPM to be available for the next version (2.5)

Configuration

- **./conf/master.conf**: master server configurations, such as Sector port, security server address, and master server data location
- **./conf/slave.conf**: slave node configurations, such as master server address and local data storage path
- **./conf/client.conf**: master server address and user account/password so that a user doesn't need to specify this information every time they run a Sector tool

Configuration File Path

- $SECTOR_HOME/conf
- ../conf
  - If $SECTOR_HOME is not set, all commands should be run at their original directory (version 2.4)
- /opt/sector/conf (available in version 2.5), with RPM installation

- #SECTOR server port number
- #note that both TCP/UDP port N and N-1 will be used
- **SECTOR_PORT**
- **6000**

- #security server address
- **SECURITY_SERVER**
- **ncdm153.lac.uic.edu:5000**

- #data directory, for the master to store temporary system data
- #this is different from the slave data directory and will not be used to store data files
- **DATA_DIRECTORY**
- **/home/u2/yunhong/work/sector_master/**

- #number of replicas of each file, default is 1
- **REPLICA_NUM**
- **2**

- #metadata location: MEMORY is faster, DISK can support more files, default is MEMORY
- META_LOC
- MEMORY

- #slave node timeout, in seconds, default is 600 seconds
- #if the slave does not send response within the time specified here,
- #it will be removed and the master will try to restart it
- #SLAVE_TIMEOUT
- # 600

- #minimum available disk space on each node, default is 10GB
- #in MB, recommended 10GB for minimum space, except for testing purpose
- #SLAVE_MIN_DISK_SPACE
- # 10000

- #log level, 0 = no log, 9 = everything, higher means more verbose logs, default is 1
- #LOG_LEVEL
- # 1

- #Users may login without a certificate
- #ALLOW_USER_WITHOUT_CERT
- # TRUE

Start and Stop Sector

- Step 1: start the security server ./security/sserver.
  - Default port is 5000, use *sserver new_port* for a different port number
- Step 2: start the masters and slaves using ./master/start_all
  - Need to configure password-free ssh from master to all slave nodes
  - Need to configure ./conf/slaves.list
- To shutdown Sector, use ./master/stop_all (brutal force) or ./tools/sector_shutdown (graceful)
  - Graceful shutdown, including shutdown of part of the system (e.g., one rack) is in SVN, will be released in version 2.5

Check the Installation

- At ./tools, run sector_sysinfo
- This command should print the basic information about the system, including masters, slaves, files in the system, available disk space, etc.
- If nothing is displayed or incorrect information is displayed, something is wrong.
- It may be helpful to run “start_master” and “start_slave” manually (instead of “start_all”) in order to debug

Sector Client Tools

- Located at ./tools
- Most file system commands are available: ls, stat, rm, mkdir, mv, etc.
  - Note that Sector is a user space file system and there is no mount point for these commands. Absolute dir has to be passed to the commands.
- upload/download can be used to copy files into sector from outside or out of sector to the local file system

Sector-FUSE

- Require FUSE library installed
- ./fuse
  - make
  - ./sector-fuse <local path>
- FUSE allows Sector to be mounted as a local file system directory so you can use the common file system commands to access Sector files.

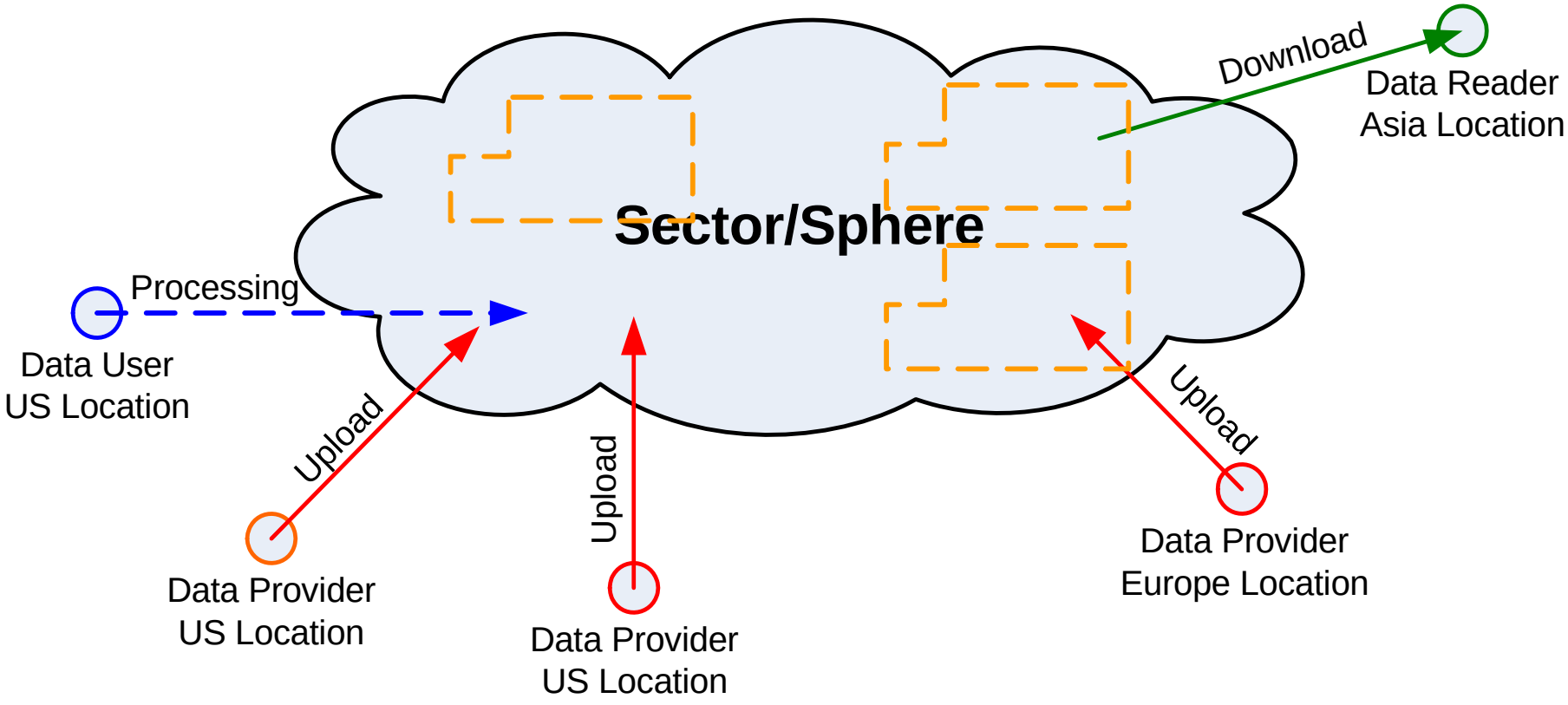
SectorFS API

- You may open any source files in ./tools as an example for SectorFS API.
- Sector requires login/logout, init/close.
- File operations are similar to common FS APIs, e.g., open, read, write, seekp/seekg, tellp/tellg, close, stat, etc.

Example Use Scenarios of Sector

- Inexpensive distributed file system: open source, commodity computers, software level fault tolerance
- Sector files are not split into blocks, thus they can be processed by other systems directly, e.g., work flow systems, grid schedulers
- Can be set up on VMs/Clouds, e.g., EC2
- Can be deployed over wide area networks
- Can be used for data sharing and distribution
  - Sector clients use UDT high speed data transfer protocol to download data from a nearby replica

Sector Data Sharing over WAN



Sector Public Cloud

- http://sector.sourceforge.net/SectorPublicCloud.html

- Test use our public Sector system to upload/download/share data

Sphere Data Processing

- Support parallel in-storage data processing
- Apply user-defined functions (UDFs) to data segments (records, group of records, files, and directories) in parallel
- Support transparent load balancing and fault tolerance

Data segmentation

- A data set consists of many files and directories
- The minimum data processing unit by Sphere is called a “segment”
- If a segment is smaller than a file, then an offset index must exist so that Sphere can use it to parse the file into segments.
  - my_data.dat, my_data.dat.idx

UDF

- int _FUNCTION_(const SInput* input, SOutput* output, SFile* file)
  - Must follow the above format
- SInput contains input data, i.e., a segment, and related information
- SOutput can be used to store the processing results
- SFile carries Sector file system information, in case it is needed by the UDF

Sphere Client Application

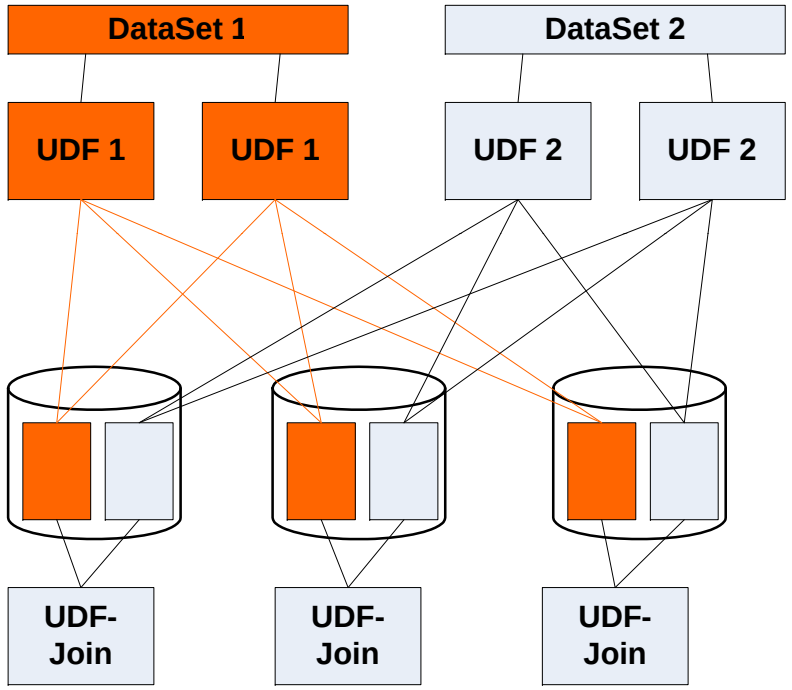
- Client init() & login()
- Specify input *SphereStream* with list of Sector files or directories
- Specify output *SphereStream* for the results
- int run(SphereStream& input, SphereStream& output, string& op, int& rows, char* param = NULL, int size = 0);
- Wait and post-process results
- Client logout() & close()

Complex Applications

- Sphere output can be the input of the next processing, therefore multiple UDFs can be applied in a sequence.
- Output can be scattered to multiple locations according to the key of each output tuple
  - Sphere can support MapReduce style applications.
- Multiple inputs can be put into directories and Sphere can process each directory as an input segment.
- Output data location can be specified when necessary, so that outputs from multiple processing can be sent to the same locations for further processing (e.g., join).

A Complex Sphere Example
Join two large datasets

- scan each data set, send data to different bucket files according the keys
- Put bucket files of the same keys on the same node
- Merge the bucket files, as they contain tuples with same keys



Conclusion

- Sector is a distributed file system
  - High performance, user space, file system level fault tolerance (via replication), support wide area networks
- Sphere supports massive parallel in-storage data processing
  - Simplified API, Transparent load balancing and fault tolerance, 2-4x faster than Hadoop MapReduce
- Open source, C++, Linux (Windows to be fully supported soon)

Thanks

- Please find more information at http://sector.sf.net
- Email me: Yunhong Gu gu@lac.uic.edu
- Open source contributors are welcome
  - 5 active contributors currently