Scalable Sage Server Architecture v0.1

Alex Leone (acleone@gmail.com)

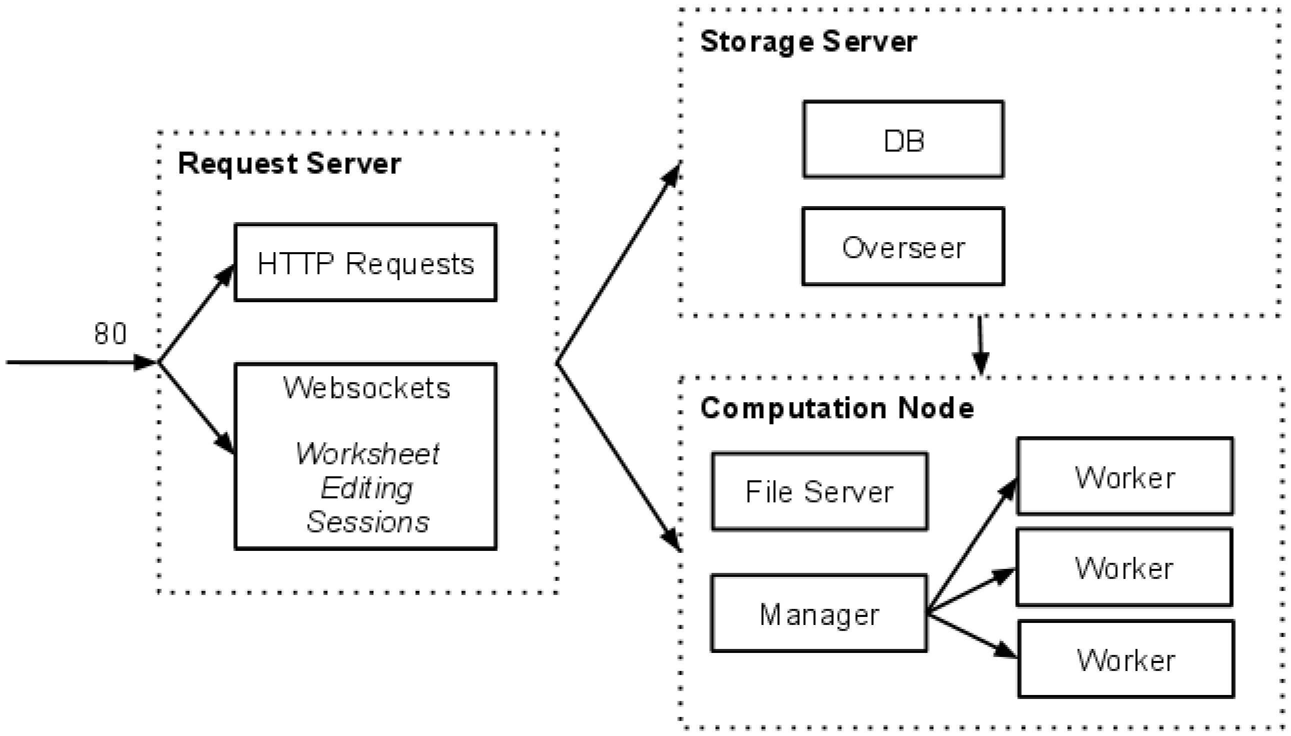
2010-12-27

Note: don't pay too much attention to the details about DB structure, http addresses, etc. These will definitely change as the design progresses.

1 What is a Sage Server?

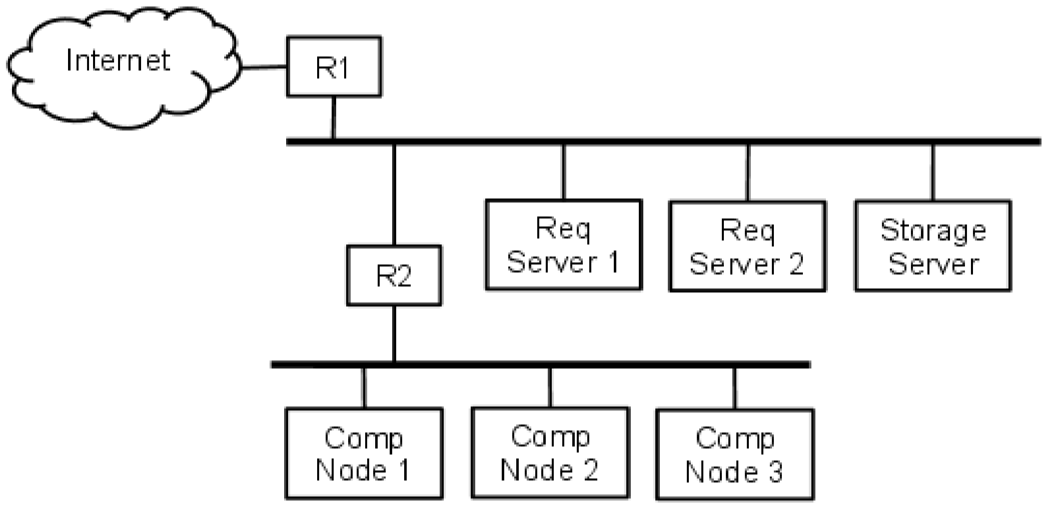
A Sage Server allows people to create, edit, and execute Sage *worksheets*.

2 Bird's Eye View



1. *Request Servers*
    a. Handles incoming HTTP requests.
    b. Handles websocket connections to *worksheet editing sessions* that synchronize clients editing the same worksheet simultaneously.
2. *Storage Server*
    a. DB for users, worksheets, groups, etc
    b. *Overseer* that assigns worksheet editing sessions to specific Request Servers and load balances (and eventually collects statistics from all the servers).
3. *Computation Nodes* (sandboxed - only incoming connections allowed)
    a. A *manager process* will start and manage *worker processes* that back a worksheet that's currently being edited.
    b. A *file server* to serve worksheet data files.

Note that this could all be on one physical computer. However, as designed, it should scale to multiple Request Servers and multiple Computation Nodes.


Internet
R1
R2
Req Server 1
Req Server 2
Storage Server
Comp Node 1
Comp Node 2
Comp Node 3

3 Action Summaries

3.1 Login, List of Worksheets, ...

For these actions, the Request Server will query the DB and return the results in html.

3.2 Editing a Worksheet

This is the heart of the Sage Server. Editing a worksheet involves both cell editing and cell execution. This involves:

1. Synchronizing the cell inputs and outputs for all clients with the worksheet open.
2. Executing cells and persisting the python process backing the worksheet.
3. Saving worksheet revision history.

3.2.1 Editing Sessions - Synchronizing Clients

To synchronize clients, each open worksheet has a corresponding *worksheet editing session* on one of the Request Servers. The Overseer ensures that a particular worksheet will have only one editing session open, and any additional clients will be redirected to that Request Server when opening the worksheet.

Clients communicate with the editing session with a Websocket (or Websocket emulator [d1]).
A Websocket allows the Request Server to send messages back to the client whenever, eg when computation output is available or another client edited a cell.

Clients synchronize cells by sending messages when an action occurs, such as adding a new cell, editing code in a cell, etc. The server will keep a shadow copy of the worksheet cells and update all clients, similar to Differential Synchronization over the whole worksheet. This will all be very similar to Google Docs.

3.2.2 Editing Sessions - Executing Code

The fun begins when a client evaluates a cell for the first time. The editing session:

1. Asks the Overseer to assign the editing session a Computation Node to run the worksheet on.
2. Asks the Storage Server to copy worksheet data files to the Computation Node.
3. Establishes a connection with the *manager process* on the Computation Node, and is assigned an initialized *worker process* that will do cell executions for the worksheet. The editing session then starts the computation in the worker process when the Storage Server has finished copying the data files for the worksheet.

This whole process should happen as quickly as possible to reduce perceived slowness of the server.

The manager process encodes output from the worker process into *messages* that are sent back to the editing session. For example, executing ‘print “Hello World!”’ would generate a Stdout(“Hello World!”) message. Other messages include Stderr, Html, Json, etc, which can be handled differently on the client side, depending on the message type. For example, Stderr messages could be displayed in red.

The worker process has multiple threads:

- The main thread runs the worksheet cell computations.
- A send/recv thread that sends and receives messages over a socket to the manger thread. Stdout and Stderr messages are encoded by the manager, but any other special messages (Html, Json, etc) are sent via this thread.
- A secondary computation thread that does completions, docstring lookups, etc.

3.2.3 Editing Sessions - Saving Revision History

For now, the "Save" button will update the DB. The editing session contacts the Storage Server, which sends a list of data file hashes for files currently in the DB to the file-system server on the Computation Node. The computation node then compares the current data files to the list, and sends any changed files. Files no longer needed in the DB are deleted.

In the future, there will be something along the lines of Google Docs, with a revision saved every time a cell is executed? Anything before the latest revision would be stored as a diff to save space.

3.3 Viewing a Worksheet

Viewing a worksheet does not cause an editing session to be opened. The Request Server serves the worksheet from the DB.

4 Implementation

4.1 Storage Server

4.1.1 Database

The database is mongoDB:

```
db.users:
{ _id: “john.doe”,
  pwHash: “873a76d87...”
  nick: “John Doe”,
  email: “john@example.com”,
  isAdmin: false,
  groups: [gid1, gid2, gid3] }

db.groups:
{ _id: gid1 (ObjectId?),
  name: “Math 414 wi2010”,
  users: [{u: uid1, perms: 1}, {u: uid2, perms: 0}] }

db.worksheets:
{ _id: ws_id1 (ObjectId?),
  title: “My Worksheet”,
  mname: “my_worksheet”, // module name for importing
  cells: [
    { _id: 3,
      type: “sage”,
      in: “print ‘Hello World!’”,
      out: [
        {t: “stdout”, data: “Hello World!\n”},
        {t: “stderr”, data: “Unhandled Exception...”},
      ] },
    { _id: 5,
      type: “html”,
      in: “<div>Hello World!</div>” }
  ],
  owner: uid,
  uPerms: [ {u: uid, perms: 7 } ],
  gPerms: [ {g: gid, perms: 4 } ],
  editingSess: {ip: “123.123.123.123”, host: “rs1.sagenb.org”}
}

db.files (gridFS) + ws_id: ws_id1
```

A few things to note:

1. db.users has the groups array for what groups the user belongs to, and db.groups has a users array for users in the group. This is because we need fast lookups for *does this user have permission to view this worksheet?* and *what users are in this group?*.
2. Permissions (db.groups.users.perms and db.worksheets.uPerms.perms) are bit-flags. 0x01 means can edit, 0x02 means can delete, 0x04 means can change title, etc. To check for permission, just use $gt (greater than).

Why use a nosql database over a row-oriented database like mysql?

1. Consider implementing the worksheet storage in mysql - there would need to be some way to store the cells array. This would probably just be a string with cell id's that would order the cells correctly. The object representation with an array of cells fits the data much better.

```
worksheets table:
  id: 42,
  …
  cells: "2,14,11"

cells table:
  id: 2,
  in: (binary blob)
  out: (binary blob)
```

2. Writes are common.

4.1.2 Storage Server Interactions

The storage server has two open ports:

1. [in] port ____, mongod (the database server)
2. [in] port ____, the Overseer
3. [out] Overseer connections to Computation Node File Servers to copy and sync worksheet data files.

4.1.3 The Overseer

The overseer has the following request-response patterns:

- `getEditingSession(ws_id)`
  - returns null if there is no editing session open for that ws_id
  - returns {ip: "123.123.123.123", host: "rs1.sagenb.org"} if there is an editing session.
- `getOrStartEditingSession(ws_id, ifNull)`
  - Called by a Request Server when a client starts to edit a worksheet. The Request Server will start an editing session if one doesn't exist yet.
  - param ifNull: if there isn't an editing session open, start one at this ip/host. Example object: {ip: "123.123.123.123", host: "rs1.sagenb.org"}
  - returns {ip: "...", host: "...", start: true/false}, where start is whether the requesting node should start an editing session (i.e. the Overseer just assigned the editing session to the requesting node).

- `assignCompNodeAndCopyDataFiles(ws_id)`
    - Called by a Request Server when the editing session needs to eval something for the first time (the editing session hasn't been assigned a worker process yet). This call will also start copying the data files for worksheet to the computation node.
    - returns {ip: "123.123.123.123"}, the ip for the computation node, and sometime later returns a success or failure message for the file copy.
- `saveDataFiles(ws_id, comp_ip)`
    - Called by the Request Server when the worksheet should be saved. The Overseer will request file changes from the Computation Node, and save the changes to the DB. The request server will write changes to the worksheet object directly into the DB.

4.2 Request Servers

The Request Servers handle client connections. There are two distinct connection types:

1. HTTP Requests
2. Websocket connections to an editing session.

4.2.1 HTTP Requests

Urls:

- / -- home page
- /ws/ -- list of worksheets
- /edit/(ws_id)/ -- edit a specific worksheet
- /edit/(ws_id)/fs/(path_to_file) -- data file for a specific worksheet, pulled from the computation node (or DB if the editing session hasn't eval'ed anything yet).
- /edit/(ws_id)/cfs/(cell_id)/(path_to_file) -- generated data file for a specific cell (eg a plot image). Cell Data files are deleted when the cell is re-evaluated.
- /view/(ws_id)/ -- view a specific worksheet
- /view/(ws_id)/fs, cfs … -- data files like above, except pulls from DB instead of the computation node where the worksheet is currently running.

Most HTTP requests are DB queries returned in html. Note however the /edit/(ws_id)/fs and cfs paths, which pull the files from the computation node where the worksheet is currently running.

4.2.2 Websocket Messages

Sent:

- cell editing and synchronization. Differential Synchronization ala Google Docs.
- "eval this cell"
- "eval this interact"
- some way to sync all client interactive controls, ie sync all the jmol or matrix viewer views across clients.
- get docstring
- get source
- get completions
- Stdin messages
- Save the worksheet
- Save and Quit
- Discard and Quit

Received:

- output message to append to cell id x
- docstring
- source

- completions

4.2.3 Editing Sessions

When a /edit/(ws_id)/ HTTP request comes in, the Request Server finds out where the editing session should be by querying the Overseer (or checking to see if the editing session is open on this computer). If the editing session is on a different server, then respond with a HTTP redirect to the other server (this computer can't serve the edit html+js because subsequent HTTP requests for plot images, etc all need to be directed to the server that is hosting the editing session for the worksheet). If the editing session is on this server, then serve the html+js for the editing page.

The js then opens a Websocket to the Request Server (which should be the one hosting the editing session or else the client would be redirected when requesting the html+js). The client will then ask for the contents of the worksheet (essentially the worksheet's json).

When a client evalutes a cell, the editing session empties the cell's output message array and sends the cell's input contents to the Computation Node in a Exec message. The computation node will start sending back output messages that go into the cell's array of output messages. The editing session will send the messages back to clients. If the cell's output gets to large, the output will be saved to a "full_output.txt". However, this is handled by the Computation Node, so the editing session will just get a FullOutput(...) message and no more output messages.

Clients should ack output messages so the editing session knows that the client has all the output messages for the cell.

Clients disconnect from the editing session with "Save and Quit" or "Discard and Quit". If the editing session loses a connection without the closing message, then the editing session will stay alive for some timeout. Perhaps there should be an "email when done" that's only available to special users, that leaves the editing session open with no timeout for long-running computations.

4.3 Computation Nodes

The most important part of the Computation Node is security. Because Computation Nodes run user code, we will assume that the Computation Node has been compromised and a user has gained root access. Therefore:

1. The Computation Node has no access rights to the Storage or Request Servers.
2. The Computation Node should not be able to start connections with the other Servers.

The Computation Nodes should be on their own network, and all other servers should block incoming connections from any address in the network (see the figure at the beginning of the document - the Request and Storage servers should block packets with MAC address equal to R2).

If a node is compromised, then only the running worksheet data on the node is insecure. Because the nodes cannot access the Storage Server, the compromised node can't read all the user data or something similar.

4.3.1 Computation Node Interactions

- getWorker()
  - Called by the Request Server when it needs a worker process. The manager will pick a worker from the pool of idle workers. From then on, any messages over this connection will go to the worker if not handled by the manager, and any output from the worker will get sent.
- copyFilesIntoDir()
  - Called by the Storage Server when an editing session on the Request Server is about to exec something for the first time, and the worksheet data files need to be copied to the Computation Node.
- getFile(http_headers)
  - serves /edit/(ws_id)/fs... requests.

Messages directed at the worker process:

- chdir(dir)
- Exec(code, options)
- Interrupt()
- Kill()

Stdin(bytes)

5 Misc

[d1] Websocket Emulator - if the client's browser does not support websockets, then it will be emulated by flash, long polling, etc. See http://socket.io/ for a good example.