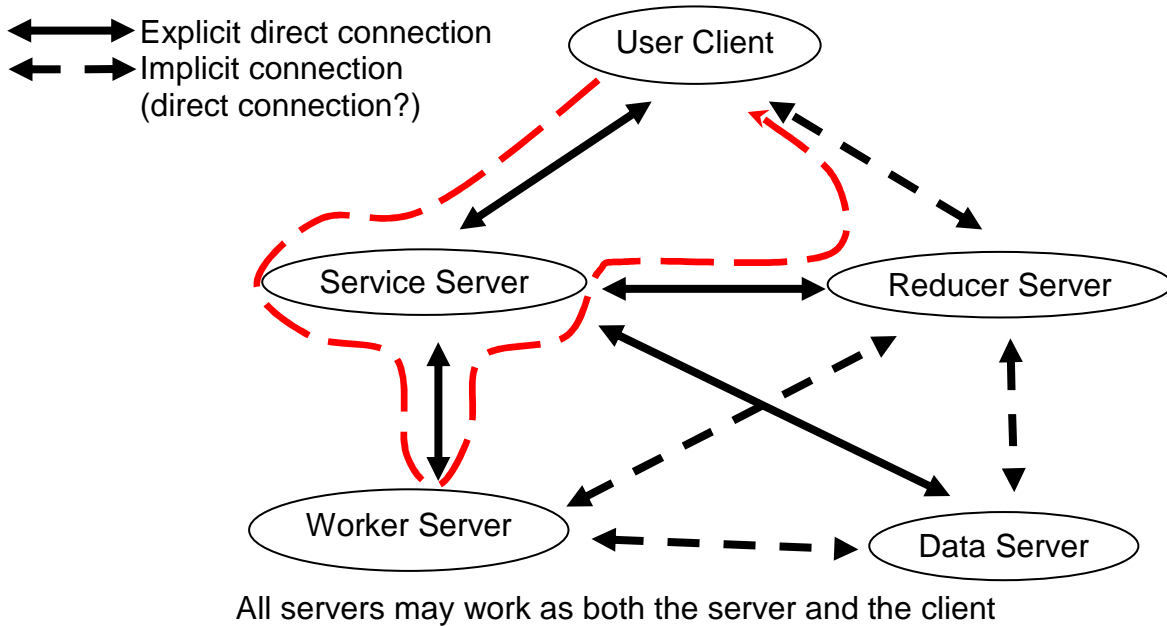


Multi-tier of PARK Architecture

The user connects the service server explicitly, and may connect the reduce server implicitly, but does not connect the working servers and data server directly.



1. All the servers have the logging system to store the request information and error message while processing those requests.
2. Only the reduce server stored the final results in its archive system. The final result has the associated key (jobID?) so the user can access it through service server. The data stored in worker server will be removed automatically [Do we need the direct communication between the user and the reduce server?].
3. Both the service server and reduce server have the message queue.
4. The service server and reduce server have the database to manage its information. The worker server and data server may or may not use the database to manage its information.

Notes: Do we need to separate the service into two parts: one for the local work, and another for the communication, for example, sending the working request to another working node directly? This can remove some restrictions of map-reduce pattern, and remove the bottleneck of the communication, for all the communications will be through the service server in map-reduce pattern. I could expect that will benefit for the more general cases, such as for GA, and that implementation will become a fully distributed and asynchronous system. However, that implementation is highly risky, and will be much more complicated.

Database Tables:

I assumed that the database has the fixed number of column, and support the array. If the database doesn't support the array, all the array fields may be stored in a separated database with the duplicated ID.

NodeDatabase: (Hardware information for working nodes)

Id	integer	ID of the node
Node_name	string	alias name
full_name	string	full name of the
ip	string	ip address (may not needed)
cpu_number	int	number of cpus
max_process_number	int	max. number of processes for the same group
mem_size	int	memory size in MB (total memory size for shared memory)
file_size	int	file system size in MB (0 means diskless node)
ssh_key	string	ssh key for the user to run the service
user_name	string	user name to run the service
service_port	int[]	ports that accept the request and works as a working node
reduce_port	int[]	ports that accept the request and works as a reduce node

ServicesDatabase: (Software information for working nodes. Service must be the executable program or functions)

Service ID:	integer	ID, primary key
Service type:	integer	(for future use)
service_name:	string	name: the user may use this name to make use the service
home_dir	string	home directory for the service
hosts	string[]	the names of the nodes that provide this service

ProjectDatabase: This database is managed by the service server administrator.

ID:	integer, primary key,	ID of the project
Name:	string , unique, can be used as primary key	name of the project
Short_description:	string,	short description of the project
Long_description:	string	details description of the project
created time:	time (float),	the time that the project is created
creator:	string	user name that create the project.
	# only the creator can stop the project	
resource-limitation:	resource limitation for the project.	
max_priority	integer	The highest priority that can be assigned to the job
max_group:	integer	max number of groups within the project
max_cputime	float	max total cpu time(s) for the project
max_filesize	float	max. total size (MB) of the files stored in archive
max_jobcputime	float	max. cpu time (s) for each job
max_jobmemsize	float	max. memory size (MB) for each job

max_jobfilesize	float	max. file size (MB) for each job (zero or negative number means no limitation)
num_group:	integer	number of active groups within the project
num_cputime	float	total cpu time(s) for the project
num_filesize	float	total size (MB) of the files stored in archive
user_members	string[] / integer[]	names or IDs of the user members # only the user member can join the project when the project is active. # the joined user can create a group, register as the listener for the groups # within the project, and query information about the project.
Activity:	bool	whether the project is active or not.
active_groups:	string[] / integer[]	names or IDs of the active groups

The system has an anonymous project, where any registered user can use it.

UserDataBase: This database is managed by the service server administrator.

ID	integer	ID of the user, the primary key
Name	string	user name, unique, can be used as the primary key
Email:	string	Email address, send email to notify that the job is completed or fails
ssh_key:	string	SSH public key, used for authentication # more fields may be needed for the authentication
contact_information		optional

ActiveUserDataBase: the user that connect to the server.

ID:	integer	user id
Host:	string	host name
Port	string	port used for the connection

GroupDatabase: managed by the creator, the user request.

ID:	integer	primary key
Name:	string	name of the group, unique within the project
Reduce:	string	name of the reduce function
Reduce_init	string	string to initialize the reduce object
Map:	string	name of the map function
Creator:	string	user name of the creator
Create_time	Time	time that the group is created.

Active_users	integer[] / string[]	ids or names of the users that are working on.
register_users	integer[] / string[]	ids or names of the users that are watching on.
Data_source	integer[]	ids of the data source used for the group.
Jobs	integer[]	ids of the jobs of the group

num_cputime	float	total cpu time(s) for the group
num_filesize	float	total size (MB) of the files stored in archive for the group

JobDatabase:

Jobid	integer	primary key
jobname	string	job name
project	integer/string	the id or name of the project
group	integer/string	the id or name of the group
user	integer/string	the id or name of the user submit this job
arrive_time	Time	the time that the job arrives in the server
priority	integer	the priority for the job
status	integer	status of the job: (init, scheduled, waiting, running, finished, killed, abort)
start_time	Time	the time that the job start to run
cpu_time	Float	the total cpu time for the job
file_size	Float	the size of archive file
running_node	integer	ID of the working nodes to run this job
datafittingID	integer[]	ID of the data fitting information
datasourceID	integer[]	ID of the data source

DataSourceDatabase: (remote data is fetched first, and saved in the local file system. The local file can be shared by all the projects and the users.)

ID:	integer	
Source:	string	The url of the remote data source
Local:	string	The url of the local data source
fileType:	integer	ID of the file type
File_size	float	size of the file (MB)
Userid	integer	the user to fetch the data, this is also the own of the file
fileLeaseTime	float	Once the owner's group is inactive, this file will be deleted after the lease time in second.
isLocal	bool	whether it is locally available

DataFittingDatabase:

To my knowledgement, the data structure must be known before the database can be created. In order to make this DataFittingDatabase as generic as possible, it just provides the links between the model database, optimizer database and result data base. For optimizerDatabase could be simple, it can be merged in this DataFittingDataBase. When query the DataFittingDatabase, the real model database and result data will be selected from the information provided by modelType field.

jobID	integer	id for the job
modelType	string	type of the model (used to locate the model database, and result database)
modelID:	integer	ID for item in modelDatabase
OptimizerID:	integer	ID for item in OptimizerDatabase
ResultID:	integer	ID for item in ResultDatabase

OptimizerDataBase:

ID	integer	
OptimizerName:	string	name of optimizer
Parameters:	string	parameters to start the optimizer

ResultDataBase:

ID	integer	
JobID	integer	jobID of the job that generates this result
Chisq	float	chisq for the data fitting
archivehost:	string	the host name for the archive result data
#archiveport:	string	the port for the archive result data
archiveurl:	string	the url (file name) to access the archive result data

ModelDataBase: should be based on the real model

For Reflectometry model with multi-layers: each layer has a layer name, the initial value and final value. Constraints and dependency are not considered.

ReflModelDataBase:

modelID	integer
postion	integer
name	model_parameter_name
initial_value	Float
final_value	Float
fixed	Bool

SansModelDataBase:

modelID	integer
name	model_parameter_name
typename	name of type, such as sphere, cylinder, ...
initial_value	Float[]
final_value	Float[]
fixed	Bool

do we need the databases to specify the constrains and dependency, and also for the multiplexor and uniplexor? Do we need an object-orientated database?

The reduce server may send request <get jobid='jobID'> related information<get/> to access the original result data for the given job. The worker server will decide whether to recalculate or store the original result data. If the original result data are saved in the working nodes, it will be saved for a while, and be automatically deleted later. The result data should be stored in the archive system in the reduce server. The user can access these data through the service server or connect the reduce server directly.