PFSsim Tutorial

Yonggang Liu (myidpt@gmail.com)
Version 0.2

System Installation

System prerequisite: Linux 32-bit system (64-bit system sometimes has error when running OMNeT++).

1. Download the latest version of OMNeT++ from http://www.omnetpp.org/, and follow the instructions to install it.
2. Download DiskSim-4.0 from http://www.pdl.cmu.edu/PDL-FTP/DriveChar/disksim-4.0.tar.gz and extract it to /disksim_path/
3. Download the latest version of PFSsim from https://github.com/myidpt/PFSsim and extract it to /github_path/
4. Type the following command, to apply the patch to DiskSim:
   # cp -r /github_path/PFSsim/disksim/* /disksim_path/disksim-4.0/
   # cd /disksim_path/disksim-4.0/
   # make
   After this, the patched DiskSim program is successfully compiled.
5. Create a new OMNeT++ project by doing:
   Type command “omnetpp” to start OMNeT++.
   Select: “File → New → project...”, extend the “OMNeT++” folder and select “OMNeT++ Project”. Click “Next”.
   Name the project as any name you like. Click “Next”.
   Extend “Single directory project, for small simulations”, select “Empty project” and click “Finish”.
6. Import the OMNeT++ component to the project by doing:
   In the OMNeT++ application, right-click the project name in the navigation panel, and select “Import...”.
   Extend the “General” folder and select “File System”, and click “Next”.
   Browse to /github_path/PFSsim/omnetpp/, and click “OK”.
   Check the “omnetpp” folder in the left window, and click “Finish”.
7. Build the OMNeT++ component by doing:
   Right click the project name in the navigation panel, and select “Build Project”.

Running the System

Suppose the patched DiskSim system is compiled and under the directory “/disksim_path/disksim-4.0/”, and the OMNeT++ component is successfully built.

Note: If you have never run the PFSsim project, please make sure the input trace files are generated and the results/ folder exist. Assume the OMNeT++ component folder is in the path “/omnetpp_project_path/”, type the following commands:

```
# cd /omnetpp_project_path/traces/
# python gen.py
# mkdir ../results/
```

To run the system:

1. First, run the following commands to start the DiskSim components.
   # cd /disksim_path/disksim-4.0/
   # ./run_pfs.sh
2. Second, run the OMNeT++ component.
   In the OMNeT++ application, right click the project name, and select “Run as … → OMNeT++ Simulation”.
3. On the pop-up window, find the “play” buttons and start the simulation by clicking one of them. The “run” or “step” buttons can show the message transmission in the graphic interface. The “fast” and “express” buttons are able to quickly generate the simulation results.
4. The PFSsim system will terminate itself when all the simulations are done.

Note:

1. The DiskSim components should be started before the OMNeT++ component starts.
2. If the program does not terminate normally, please kill the DiskSim process by typing the following command:
   # pkill physim
3. The OMNeT++ component process can be stopped by clicking the red button on the panel of the main application

interface.

4. The default traces will be run for about 4.2 seconds in simulated time.

Please send your questions to myidpt@gmail.com if you have any problem when going through this process.

Introduction of Classes in OMNeT++ Component

Folder: client/

Client

Reads the traces from the trace file, and creates a Trace object per trace. For each Trace, a qPacket is created and sent to the metadata server (Mserver) to query the layout of the requested data. The qPacket carries the layout information back and the Client writes the layout information to the Trace object. And then, the gPackets carrying the data access requests (the destination Dservers are set) are generated by the Trace object. The Client schedules to send the gPackets to the data servers (Dservers). Each time the Client receives a gPacket carrying the reply message of a data access, it redirects the gPacket to the Trace object, and the Trace checks if more data need to be accessed. When all the data accesses from the current trace are successfully done, the Trace object is detroyed by the Client. The Client reads the next trace when the previous one is done.

Configurations are in client/Clientspecs.h

Configuration Table

Name	Description
OMNET_CID_BASE	Default value is 2. The Client IDs are OMNeT++ object IDs minus OMNeT_CID_BASE.
C_TOTAL	Total number of Clients in theID simulator
MAX_APP	The maximum number of applications
CID_OFFSET	The offset of the client ID in the packet ID (Packet ID = Client ID * CID_OFFSET + localID)
C_TRC_PROC_TIME	The process time of a new trace at the client
C_PKT_PROC_TIME	The process time of a new packet at the client
TRACE_PATH_PREFIX	The prefix of the path of trace file. The prefix added by 4 digit numbers are the entire trace file paths. Eg., if the prefix of path is abc/def, the trace file paths start from abc/def0000.
RESULT_PATH_PREFIX	The prefix of the path of result file. The prefix added by 4 digit numbers are the entire result file paths. Eg., if the prefix of path is abc/def, the result file paths start from abc/def0000.

Folder: trace/

Trace

It is created by the Client upon reading a new trace from the trace file, and destroyed by the Client when all the data accesses are all done for that trace. The Trace object contains a Layout object which illustrates the layout of the requested data. The data may be stored in multiple chunks on a certain number of data servers. Each time, the Trace issues a certain amount of data requests according to the window size of the data layout. Once the reply of all the sent data requests are received, the Trace will move the window forward and start to issue the new requests.

Folder: layout/

Layout

Each Trace object contains a Layout object. And the metadata srever (Mserver class) maintains a list of Layout objects, each for one application. On the Mserver, the Layout object reads the layout information from the layout configuration file

(default path: config/layout) at the system start up, and it's information is copied to the qPacket reply when a qPacket request from Client comes. On the Client, when the qPacket reply is feed to the Trace object, its layout information is copied to the Layout object.
Currently the Layout class simulates a data layout manner as follows: the data are stored onto a designated data servers in a round-robin manner, and each data server has a chunk size, meaning in each round, the data server stores a specific size of data. Based on this information, the Trace class has a notion of window, that in each window, each designated data server is accessed once.

Folder: mserver/

Mserver

The Mserver class simulates the metadata server component in the parallel file systems. The Mserver reads the data layout information from the configuration file (default path: config/layout) at the start up, and creates one Layout object per application. When the Mserver receives the qPacket querying the layout information, it looks up the application ID in the Layout object list, and returns the layout information by writing it to the qPacket and sending it back.
In current version, only one metadata server can be simulated, and no communication is simulated between the metadata server and the data server.

Configurations are in mserver/Mserverspecs.h

Configuration Table

Name	Description
MS_PROC_TIME	The process time of qPackets on the metadata server.
LAYOUT_INPUT_PATH	The layout configuration file path. Default value: “config/layout”.

Folder: dserver/

DSdaemon

A Dsdaemon object simulates one data server daemon component on the data server. It queues the incoming gPackets from the Ethernet interface and schedule them to the local file system according to the specified strategy. The default strategy is FIFO. The queue is able to control the number of outstanding requests in the local file system by setting the DSD_QUEUE_DEGREE parameter.

Configurations are in dserver/Dserverspecs.h

Configuration Table

Name	Description
DSD_NEWJOB_PROC_TIME	The processing time of a new request on the data server daemon.
DSD_FINJOB_PROC_TIME	The processing time of a finished request on the data server daemon.
DSD_DEGREE	The maximum number of outstanding requests concurrently dispatched from the data server daemon.

LocalFS

A LocalFS object simulates one local file system component on the data server. It maintains the Cache component which simulates the memory of the local file system. The Cache component gets the page-ranges (a range of sequential pages) that exist in the cache and the page-ranges that need access to the disk. The time consumed on cache access is simulated on the LocalFS object, and the time consumed on disk access is simulated by the DiskSim simulator.
The data placement strategy on the disk can be defined by the users. The default layout is sequential layout, which means all the data from one application are sequentially stored on the disk.

Configurations are in dserver/Dserverspecs.h

Configuration Table

Name	Description
PAGE_SIZE	The size of a page in the local file system, it is equal to the size of a block on the disk.
MAX_PAGENUM	The maximum page index in the local file system. It restricts the size of the page table.
LOCALFS_DEGREE	The maximum number of outstanding requests concurrently dispatched to the local file system.
CACHE_R_SPEED	The time consumed on reading one page in cache.
CACHE_W_SPEED	The time consumed on writing one page in cache.
MAX_PR_PER_REQ	The maximum number of page-ranges one local file request can incur. This parameter is used internally in numbering page-range requests.

Disk

A Disk object takes care of the synchronization between the OMNeT++ simulator and one DiskSim instance. It initializes the TCP connection with DiskSim at start up, and always synchronizes with DiskSim through the simulation.

Configurations are in dserver/Dserverspecs.h

Configuration Table

Name	Description
DISK_ID_BASE	The base offset of the disk IDs. Used to calculate the port numbers. Default value: OMNET_CID_BASE+C_TOTAL+DS_TOTAL*3
DISK_DEGREE	The maximum number of concurrent data requests on the disk.
APP_DATA_RANGE	The maximum number of blocks one application data can take.
BASE_PORT	The lowest port number for the synchronization with DiskSim. Default value: 8800.

Folder: cache/

ICache

The base class for the cache component implementations. It provides the interfaces of reading cache, writing cache, flushing cache, getting the cached size of a given data access, etc. Note that the unit in the cache is page. The “page-range” structure is defined for recording the accessed or cached pages. It is a linked list in which each element represents a continuous page range that has the same properties, such as modified or referred.

NRU

The NRU class inherits from the ICache class, and it partly simulates the “Not Recently Used” algorithm. The algorithm manipulates the page table based on the “page-range” structure. Currently we have not implemented the referred flag reset clock interrupt.
The reference to NRU is:
http://en.wikipedia.org/wiki/Page_replacement_algorithm#Not_recently_used

Folder: router/

Routing

Simulates the routing/switching functionality inside a router/switch in a real network. Directs the packets to different interfaces according to the packet kinds and IDs.

Queue

Simulates the queues at the interfaces on the router/switch.

Folder: scheduler/

Scheduler

A Scheduler object simulates the scheduler in a real network. This Scheduler class intercepts the data requests as well as the data replies. Scheduling algorithms are implemented as stand-alone classes by inheriting the IQueue class, and they are plugged into the Scheduler class.

Configurations are in scheduler/Schedulerspecs.h

Configuration Table

Name	Description
SCH_ALG	The scheduling algorithm used on the scheduler. Currently it can be one of the algorithms: FIFO_ALG, SFQ_ALG.
SCH_NEWJOB_PROC_TIME	The delay on the scheduler on processing a new request.
SCH_FINJOB_PROC_TIME	The delay on the scheduler on processing a finished request.
SCH_DEGREE	The maximum number of outstanding requests dispatched to the data server.

IQueue

The base class for any scheduling algorithms. This class supports two types of scheduling policies. The first type only maintains a queue for the awaiting requests, and does not concern about the requests on the server. The second type maintains a queue for the awaiting requests, and also maintains a queue for the requests on the server, thus it is able to keep track of the request service time and server load.

FIFO

This class inherits from the IQueue class and it realizes the FIFO scheduling algorithm. This algorithm maintains a queue for the awaiting requests, and a queue for the requests on the server, thus the user is able to set up a degree to restrict the number of outstanding requests on the server.

SFQ

This class inherits from the IQueue class and it realizes the Start-time Fare Queuing scheduling algorithm. The reference is:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.1054&rep=rep1&type=pdfg
This algorithm maintains a queue for the awaiting requests, and a queue for the requests on the server, thus the user is able to set up a degree to restrict the number of outstanding requests on the server.

Folder: Packets/

gPacket

The general packet is mainly used for data transfer between entities and components. It can be used for recording the detailed time stamps when processing the data. The offset is stored to highoffset and lowoffset in order to prevent overflow of values in the offset (.msg files do not support long long type). The ID of the packet is in the “Id” parameter. The ID of the data server that serves this packet is in the “decision” parameter. The application ID is in the “app” parameter.
The general packet is also used for event triggering (the SELF_EVENT kind) in one component internally.

qPacket

The query packet is used for querying and replying the data layout information between Client and Mserver. It records the application ID, corresponding data server numbers, corresponding data server IDs and the size of the chunks each corresponding data server takes.

Spacket

The schedule-information packet is used for the inter-scheduler communications. Currently we have not used this packet in the existing scheduling algorithms.

Input and Output Files

Trace Files

The trace files are in the folder traces/. A simple python program is given to generate the trace files. Trace files are named with a common prefix with sequential 4-digit IDs, which start from zero. The Clients are also sequentially numbered from zero in OMNeT++. Thus, each trace file will be read by the Client with the same ID. The prefix of trace file paths are defined as the macro TRACE_PATH_PREFIX in Clientspecs.h.
Each line in the trace file represents a request submitted by the system client.
The format of each line in the trace files:

Dispatch Timestamp	The timestamp when the client dispatches this request. It is meaningful only when this trace does not have the synchronization flag set.
Offset	The offset of the data access, in byte.
Size	The size of the data access, in byte.
Read Flag	The read flag of the data access. 1 if it is read; 0 if it is write.
Application ID	The ID of the application that dispatches the request.
Synchronization Flag	Meaning if the dispatch of this trace is synchronized with the finish of the previous trace. If the flag is 0 and the previous trace is finished (if any), the real dispatch time is the given dispatch timestamp, otherwise, the real dispatch time is the finish time of the previous trace.

Layout File

The layout files are read by the metadata servers to set up the data layout for each application in the system. It must cover all the application IDs appearing in the trace files. There is only one layout file read by the metadata server at the beginning of the simulation.
Each line in the layout file represents the data layout for an application.
The format of each line in the layout file is:
application_ID [dataserver_ID dataserver_share] [dataserver_ID dataserver_share] ...
In each [dataserver_ID dataserver_share], the dataserver_ID means the ID of the data server that stores part of the application data, the dataserver_share means the chunk size of the data stored on this data server.
In the default data layout strategy, the data of one application is stored in a round-robin manner on the corresponding data servers, and each data server only stores the data of dataserver_share size in each round.

Result Files

The result files are in the folder results/. The result files are numbered in the same way as the trace files. Each Client will generate the result file with same IDs. The prefix of result file paths are defined as the macro RESULT_PATH_PREFIX in Clientspecs.h.
The result files contain the information of traces and the Client-Dserver data packets in the simulation. The corresponding information is written when the trace or the Client-Dserver gPacket is finished. The users are able to define the information written to the result files by rewriting the methods:

```
void Client::reqStatistic(Request *);
void Client::pktStatistic(gPacket *);
```

Mechanisms in the DiskSim Component

DiskSim is an efficient, accurate, highly-configurable disk system simulator. We patched DiskSim to enable the synchronization with the OMNeT++ component. The DiskSim component is capable of processing multiple disk access requests in parallel.

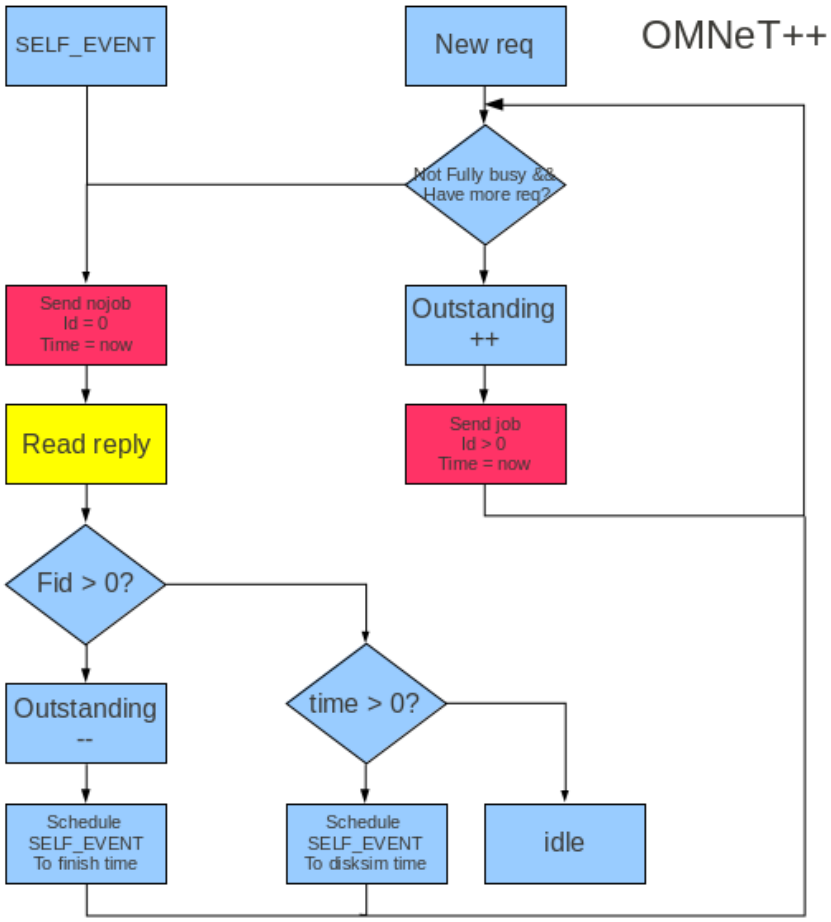
DiskSim has provided the diskmodels for some example disks, while for other disks, DiskSim has provided the DIXtrac program to generate the diskmodels from real disks. For details, please check http://www.pdl.cmu.edu/DiskSim/.

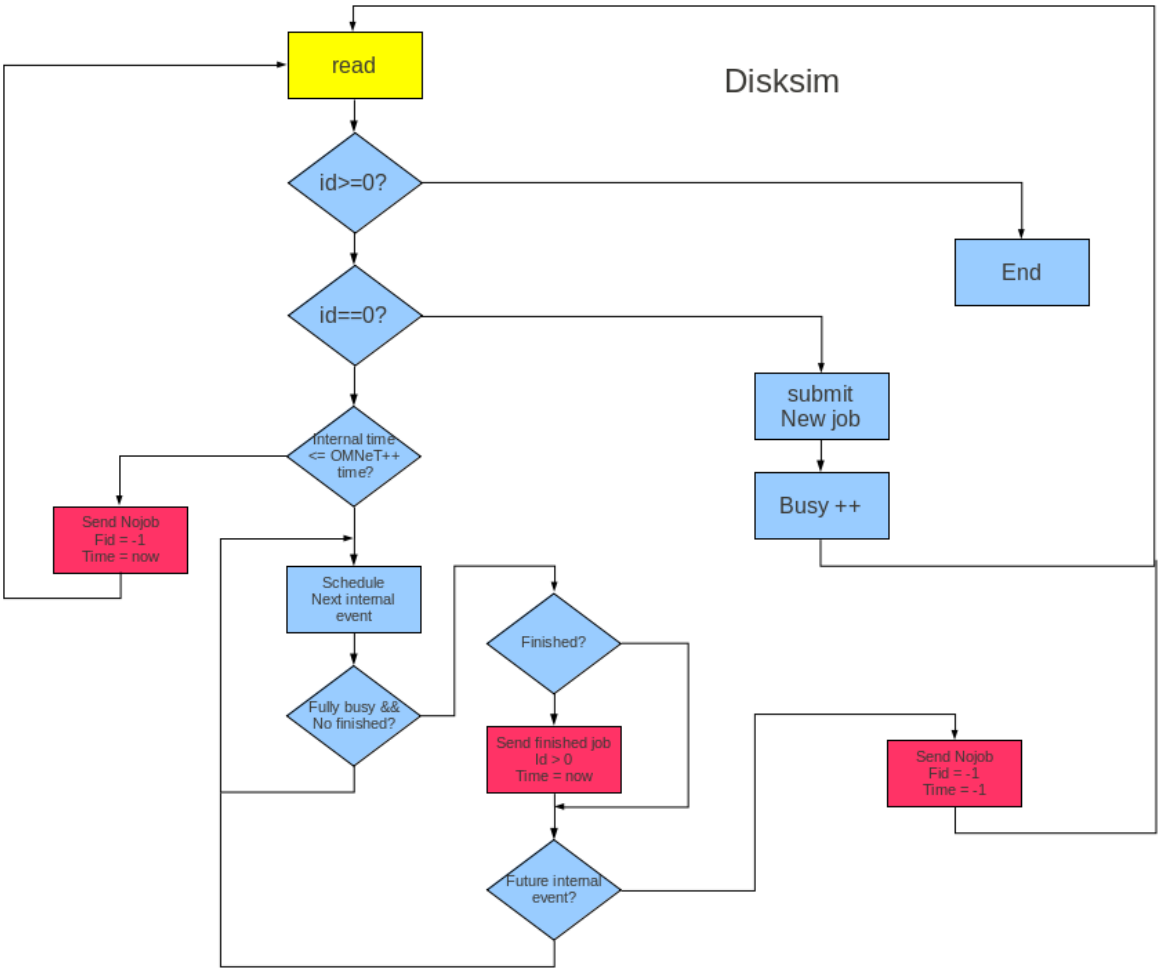
Configurations are in the header file physim_driver.h under disksim-4.0/src/.

Configuration Table

Name	Description
BLOCK	The size of a data block.
DEGREE	The maximum number of outstanding data requests on the disk driver.
SYNC_SIZE	The size of the synchronization messages.

State Machines of Synchronization between OMNeT++ and DiskSim




Disksim
read
id>=0?
End
id==0?
submit
New job
Busy ++
Internal time
<= OMNeT++
time?
Send Nojob
Fid = -1
Time = now
Schedule
Next internal
event
Finished?
Fully busy &&
No finished?
Send finished job
Id > 0
Time = now
Send Nojob
Fid = -1
Time = -1
Future internal
event?