SAND2008-2832C

Development of a Parallel Transfer Model for OPNET

John Sherrell
Electrical and Computer Engineering
Kansas State University
Manhattan, KS
Email: jms7373@ksu.edu

Jason S. Wertz
Network Design and Operations
Sandia National Laboratories
Albuquerque, NM
Email: jswertz@sandia.gov

Don Gruenbacher
Electrical and Computer Engineering
Kansas State University
Manhattan, KS
Email: grue@ksu.edu

***Abstract*—We introduce a Discrete Event Simulation (DES) software model for a generalized application that transfers bulk data over parallel TCP streams. We discuss the features and design philosophies of OPNET®, the software environment where our model was developed and where it runs. An overview is given of real-world parallel transfer (PT) applications and we arrive at a basic set of parameters that a PT model should support to be sufficiently general. We present our model design and breifly discuss its runtime behavior. Finally, some preliminary results are presented that compare simulated performance of parallel transfers on a specialized high-performance network to actual test data.**

I. Introduction

A common performance optimization for network data transfer applications is to move a single logical transfer in parallel over multiple TCP streams. This technique can dramatically increase aggregate throughput even when the end-to-end network capacity remains fixed. The technique is effective because TCP is often a significant performance bottleneck in modern networks []. There are several reasons why parallel TCP streams can improve overall performance: they compensate for poorly tuned TCP parameters, they increase fairness for high-latency flows, they reduce the time required to recover from packet loss, and they provide an opportunity for application-controlled load-balancing over parallel paths on the underlying network.

We desire a tool that can predict parallel transfer (PT) application performance from a wide range of relevant application, network, and storage parameters at a substantially lower cost than testing each scenario on a live network. An analytical model for aggregate throughput of a parallel stream data transfer is proposed in [], but most other related studies have been empirical in nature. This paper introduces a Discrete Event Simulation (DES) software model of a generalized parallel file transfer application. We first discuss real-world PT applications to form a basis for the parameters the model should suport to be sufficiently general. Next, we introduce OPNET®, the modeling software used to develop and simulate our model. We will emphasize the flexibility it provides in modeling an enormous range of scenarios and system parameters. Finally, a description of the model's design is provided along with some preliminary results.

A. Parallel Transfer Applications

There exists a wide variety of real-world PT applications with diverse designs. Parallel File Transfer Protocol (PFTP) [] opens persistent, FTP-style sessions, and allows users to set the *block size* and *stripe width* for a transfer. The block size is the amount sequential file data that will be accessed by a single socket or filesystem operation. The stripe width is the number of parallel streams used by the transfer. In PFTP, file data is *striped* over the parallel streams, so that if the stripe width is N, each stream will seek $N-1$ blocks through the file between each of its disk operations []. Finally, PFTP enforces a lock step block transfer protocol, where each stream must wait for an application-layer acknowlegement after every block it sends before sending the next block. MPSCP is similar to PFTP, but it benefits from increased efficiency because it does not use a lock-step block transfer protocol and because its striping procedure is more flexible, with each stream always transmitting the next available file block. Other PT applications include simple download manager applications that run on plain FTP and Hyper Text Transfer Protocol (HTTP). These applications tend to not use the striping procedure mentioned above, instead assigning one large, contiguous segment of the file to each stream. Neither protocol employs a lock-step, and HTTP transfers have no real overhead to speak of.

In general, a PT model should allow specification of the following parameters:

- the per-session, per-transfer, and per-block overheads,
- the stripe width and block size,
- the method for assigning blocks to streams, and
- any required synchronization (like lock-step protocols).

The per-session and per-transfer overheads should have a negligible effect on performance for large file transfers, but might dominate total delay for many small transfers.

B. OPNET Modeler

OPNET is a software environment used to model and simulate networks. Models in OPNET are hierarchical: network models consist of instances of node and link models, and node models consist of instances of process models. Each process model defines a finite state machine (FSM), the behavior of which is fully specified in C or C++ source code. Process

model code frequently calls library functions provided by OPNET, called kernel procedures (KPs), which provide useful services like packet manipulation, interprocess communication, and random sampling of statistical distributions

An important feature of OPNET is that it ships with an extensive standard model library, encompassing a wide variety of network hardware and protocols. Thus, if we were confident in the validity of our PT model, we could collect an amazing array of useful data. We could, for example, simulate PT application performance for different TCP congestion control algorithms, various background traffic types, increased layer-2 MTUs, various QoS policies in intermediate routers, or many other relevant parameters.

II. Design

The PT model is designed at the traditional application layer in the TCP/IP stack, which is different from the standard OPNET applications. As Figure 1 shows, the PT model interacts directly with TCP, while the standard OPNET applications reside above a couple of additional abstractions: the TPAL and GNA layers. The primary feature of the TPAL layer is the transport protocol-independent API it offers to applications. However, TPAL requires applications to use server names instead of IP addresses, and TPAL semantics also restrict each application type to just one listening (incoming) TCP port. Both of these restrictions would prevent a proper PT application implementation, because the application must be able to determine which port and interface to bind each listening socket to. Applications utilizing the GNA layer are intended to implement only client logic and to generate traffic patterns by sending commands such as `[respond with 3 packets]` to a generic, shared server. Obviously, this approach is too restrictive for the logic-dense PT model, which requires specific server behavior. One useful service provided by the GNA layer is the configuration paradigm of usage profiles. With this approach, configured instances of applications are defined, and then profiles are defined that *use* the application instances in some pattern at configurable simulation times. This works well with typical OPNET modeling workflows, and although it is currently not supported by the PT model, it might be desirable to integrate it into the model in the future.

The PT model is implemented as a set of four OPNET process models with the hierarchical relationships illustrated by Figure 2. There are not separate client and server models. The process models underlying both ends of a PT application session are identical; one side is simply configured to wait for a connection while the other is configured to initiate the connection. We prefer the *user* label to describe the node actively opening the connection, and the *listener* label to describe the node that accepts the connection. Actually, these labels are only valid for a node in the context of a given session, because nodes can participate in multiple sessions simultaneously. In fact, all nodes are *always* listeing for connections. At initialization , each PT-enabled node will run a single `manager` process, which will spawn a single

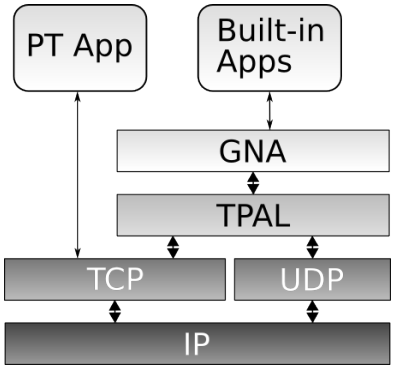

Fig. 1. PFTP striping a file transfer over four parallel TCP streams.

`filesystem` process and a listening `session` process. User `session` processes are spawned by the manager on-demand at whatever simulation times the parent node is configured to initiate a transfer session. Unlike with Unix sockets, once the listening `session` accepts a connection, the listening socket is consumed. We have created a mechanism to notify the parent `manager` when this occurs so it will immediately spawn a new listeining `session`. Sessions may overlap, so a node might be running two or more session processes simultaneously. Each session models one or more file transfers, depending on the user configuration. File transfers have an associated stripe width, which determines how many `mover` processes are spawned at both sides of the connection. Movers are paired up by their parent `session` processes. Once connected, `mover` processes transfer blocks, optionally requiring acknowlegements after each transfer to simulate the lock-step of PFTP. The rigid assignment of blocks to movers in PFTP is simulated by requiring all movers to transmit the same number of blocks. The dynamic assignment method used by MPSCP requires some communication between the sending `mover` processes and thier `session` parents. A transfer is complete when all `mover` processes are finished.

In some special scenarios, network performance might surpass local filesystem performance. Parallel streams utilizing fast new networking technologies can provide sustained throughputs too large for disks to source or sink. Thus it is important to model cases where filesystem performance is the limiting factor in a parallel transfer. Commonly, sites interested in high-speed data transfer will avoid the local bottleneck by striping files over many disks. However, PT applications that stripe block data over the network streams will likely see better performance on parallel filesystems than will PT applications that transfer large contiguous segments, which will likely generate excessive disk seeking. Thus, we desire a model that can adapt filesystem performance based on application parameters. Finally, we anticipate the need to model special storage scenarios such as retreving a file from a high-latency tape archive. For the flexibility to model all of these possibilities, we created the previously mentioned `filesystem` model. The model provides file access modeling at a high level. The `mover` processes issue block read or write requests

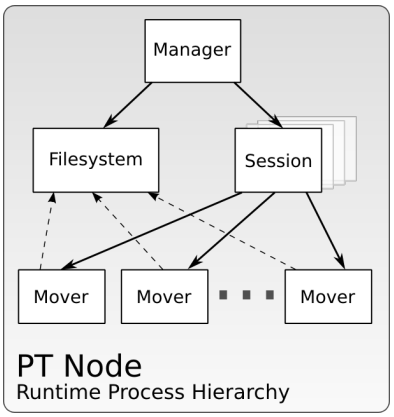

Fig. 2. The hierarchy of process models that make up the PT model.

to the node's `filestem` process. The `filesystem` process will then simulate a general file read or write delay before notifying the original `mover` process that the I/O operation is complete and it may continue working. Currently, operation of the `filesystem` model is trivially simple: it uses random samples from an exponential distribution as the delay values, where the mean of the distribution is user-configurable. The `filesystem` model needs substantial implementation and validation work to fulfill its purpose.

III. Results

We now present some preliminary results. Figure 3 plots simulated aggregate parallel transfer rate versus stripe width (number of streams) for various levels of filesystem performance. Also plotted are actual values obtained from PFTP tests on the Advanced Simulation and Computing (ASC) program WAN, which connects together the high performance computing (HPC) resources at the Department of Energy/National Nuclear Security Agency (DOE/NNSA) main weapons laboratories. The ASC WAN consists of 10 Gigabit links that span up to several hundred miles between laboratory sites in central New Mexico and Livermore, California. The PT model was installed in an existing OPNET network model of the ASC WAN to obtain the simulated results. As the plots show, the simulation results are fairly realistic up to four parallel streams, especially for larger storage delay values. At this point (around 100 Mbytes/s), we believe the real ASC hosts might be limited by filesystem performance, but this will require further investigation. As we add features to our `filesystem` model and collect more test data, we expect to obtain more interesting and definitive simulation results.

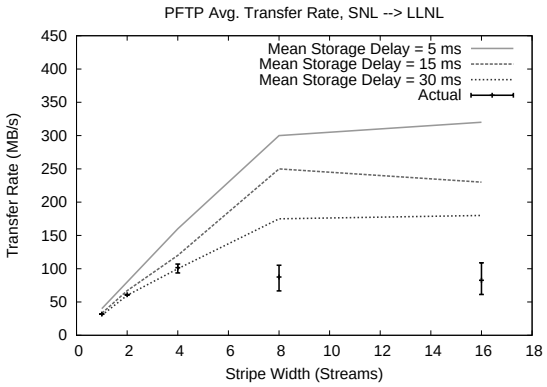

Fig. 3. Transfer rate versus stripe width for various mean storage delays. These results are preliminary.