StreamGen: A Workload Generation Tool for Distributed Information Flow Applications

Mohamed Mansour, Matthew Wolf, Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{mansour, mwolf, schwan}@cc.gatech.edu

Abstract

This paper presents the StreamGen load generator, which is targeted at distributed information flow applications. These include the event streaming services used in wide-area publish/subscribe systems or in operational information systems, the data streaming services used in remote visualization or collaboration, and the continuous data streams occurring in download services. Running across heterogeneous distributed platforms, these services are implemented by computational component that capture, manipulate, and produce information streams and are linked via overlay topologies. StreamGen can be used to produce the distributed computational and communication loads imposed by these applications. Dynamic application behaviors can be created with mathematical specifications or with behavior traces collected from application-level traces. An interesting set of traces presented in this paper is derived from long-term observations of the FTP download patterns observed at the Linux mirror site being run by the CERCS research center at the Georgia Institute of Technology.

Two different flow-based applications are created and evaluated with StreamGen. The first emulates the data streaming behavior in a distributed scientific collaboration, where a scientific simulation (i.e., a molecular dynamics code) produces simulation data sent to and displayed for multiple, interactive remote users. The second emulates portions of the event-streaming behavior of an operational information system used by a large U.S. corporation. Parametric studies with StreamGen's FTP traces applied to these applications are used to evaluate different load balancing strategies for the cluster machines manipulating these applications' data streams.

1. Introduction

Motivation. Benchmarks are a common tool for assessing the performance of distributed computational platforms and their runtime infrastructures. Evaluations may focus on systems' end-to-end performance, compare different hardware architectures or middleware implementations, or assess individual subsystems' capabilities. A classification of benchmark types for parallel systems appears in [2]. From such previous work, it is well-known that microbenchmarks or macro benchmarks like TPC-x [8] are not sufficient for providing rigorous insights into the capabilities and behaviors of distributed hardware and software infrastructures. As a result, it is common practice to test and evaluate them with actual applications [3]. A general technique used in such cases is trace-based benchmarking, which involves the creation of a parameterized model of the actual or anticipated system workload, which is then used to conduct performance evaluations of the systems under consideration.

The StreamGen workload generator. This paper describes a workload generator, termed StreamGen, which may be used to create a set of distributed services interacting via an application-level overlay network. StreamGen is targeted at distributed information-flow applications, which are composed of computational kernels interconnected with communication links structured as graphs, such as pipelines or data distribution trees. Each kernel performs computations on units of data received as inputs and produces outputs as units of data sent to other kernels. End results are output at leaf nodes and/or at intermediate nodes.

Two concrete instances of information flow benchmarks are the NAS Parallel Benchmark (NPB) [4] and SPEC HPC2002 [7], which address the domain of high performance applications. Both are composed of computationally intensive interconnected kernels that implement specified target applications. StreamGen can be used to create such interconnected computational structures, and its workload generation facilities can implement the parameter and configuration settings proscribed by their benchmark specifications. StreamGen has additional capabilities, however. First, in contrast to the well-defined goals of NPB, StreamGen does not prescribe the use of certain computational kernels, since that would make it difficult to create meaningful emulations of the wide range of applications for which StreamGen is intended. Second, StreamGen can emulate

the dynamic departures or arrivals of benchmark participants, where each participant's behavior may change at runtime, as her interests change or as the data being manipulated changes its nature.

StreamGen addresses commercial workloads in a fashion that complements transactional benchmarks like TPC-C [8] and ECPerf [9]. TPC-C is an OLTP benchmark that specifies a complex transactional workload, including initial database seeds and the amounts of work generated at the database level. The purpose of TPC-C is to compare the performance/value offered by different hardware/software systems. ECPerf is similar, but is designed to benchmark alternative implementations of J2EE servers. Both benchmarks are well suited for OLTP applications, but neither defines or operates with the dynamic behaviors of today's distributed information-flow applications. StreamGen's role in this context is to permit users to create distributed information flows and kernels that operate on those flows, sample kernels ranging from the complex business rules applied in operational information systems [11] to the simpler event filters or fusion expressions used in wide area event system [10].

The StreamGen workload generator has multiple components. Its workload generation tool, termed StreamGen-Gen, is derived from HttPerf [1]. This tool permits end users to generate workloads in which multiple data sources can impose loads on a single sink, or where a single source can impose workloads on multiple sinks. The link topologies used are defined by end users, but the APIs exposed by StreamGen permit a linked node to act as a sink, source, or both. The middleware used to implement such links is also determined by end users. The specific infrastructure used in this paper's evaluations of StreamGen's utility is the ECho publish/subscribe middleware, which is focused on interactive high performance applications like real-time collaboration.

An important attribute of StreamGen is its ability to generate and use typed information flows, i.e., flows that are comprised of typed data units. The efficient structured binary data representations [16] used for this purpose permit diverse computational kernels to interact via a single, uniform data representation. This is important for two reasons: (1) to conveniently implement the many diverse data formats typically found in heterogeneous distributed applications, and (2) to explicitly model (and measure) the data conversion actions necessary across different application components, the latter important both in the domain of large-data applications like scientific collaboration and commercial codes like operational information systems.

Dynamic workload behaviors. StreamGen explicitly supports *dynamic* information flow behaviors, via a library for traffic generation and a facility for traffic replay from captured traces. The particular traces used in this paper were obtained from analyses of FTP traffic to the Georgia Tech Linux FTP mirror server. This site houses many different Linux distributions, including multiple RedHat, SuSE, Mandrake, and Debian Linux releases, with the combined repository resources supporting close to 2 TB of data. The site is updated whenever new releases appear. It is used by end users to download or update Linux sources and binaries, and it collaborates with other mirror sites to maintain consistent data across multiple mirrors (note that there are 53 major Linux mirrors in the US). The popularity of the Linux operating system makes the GT/CERCS mirror site a popular destination. The total traffic experienced by the site is substantial, averaging 2500 hits/hour and peaking at 300,000 hit/hour during periods of high activity. The total traffic traces collected over two years are almost 13GB in size. Previous work [14] presents an algorithm for reconstructing user FTP sessions from low-level ftp traces.

Apart from providing us with sample dynamic traces, two outcomes of mirror measurements relevant to this paper are that this site experiences (1) high degrees of concurrency in file retrieval and (2) substantial burstiness of retrievals, the latter correlated with the arrival of new Linux releases. Namely, this information-flow-based application experiences levels of burstiness for relatively long-lived flows (i.e., entire Linux downloads) that are similar to those seen for web sites that contain dynamic information, as with the Mars Rover data, results of sports events, etc. We hypothesize that this is also the case for scientific data repositories accessed non-programmatically (i.e., directly accessed by end users) or programmatically (by mirror sites), thereby making our work relevant to the broad domains of high performance and grid computing.

Using StreamGen. This paper uses StreamGen and the dynamic traffic traces measured for the Linux mirror to better understand two classes of distributed applications. The first class concerns interactive scientific collaborations in which multiple end users define and use their own views on dynamically generated, shared simulation data. The StreamGen implementation generalizes the SmartPointer collaboration system first presented in [5], where output data is streamed from a large parallel computational science code (molecular dynamics) to a series of annotation and visualization services (see Figure 3). An individual end user can dynamically change this data, by way of various service components, to customize data representation and display to fit his or her current needs. End users range from high-end clients using display facilities like Immersive Desks to clients with small screens and very limited computational (e.g., handheld devices) and communication (e.g., wireless links) abilities. Parametric studies performed with StreamGen evaluate different load balancing strategies

when clients are associated with visualization services, using multiple visualization servers.

The second class of applications evaluated with StreamGen concerns operational information systems. The StreamGen implementation captures some of the dynamic behaviors observed in and relevant to such systems. Here, a major airline carrier uses a distributed system to capture, analyze and store different business events as well as make business decisions and disseminate such decisions to their office and airport terminals. The StreamGen setup exercises enterprise servers with two distinct traffic patterns. In one pattern, thousands of clients are connected to these servers, each with a low rate of requests per second. This emulates airport terminals, where requests are generated as fast as operators can type. In the other pattern, a few clients connect to the enterprise server, each generating a high rate of requests per second. This is typical of e-commerce web front-ends connecting to legacy back-end systems. Requests could be processed by one server, or can result in a chain of internal requests propagating through the enterprise system and generating multiple database requests.

In the remainder of this paper, Section 2 describe the architecture and implementation of StreamGen. Section 3 presents the two application codes generalized with StreamGen. Section 4 presents experimental results attained with these applications on representative hardware platforms. Conclusions and future work appear last.

2. Architecture and Implementation

StreamGen emulates distributed applications structured as distributed services operating on data streams. Services merge items from multiple streams, apply data transformations, personalize or annotate data items, or apply business rules to them. The links between services are described by directed acyclic graphs (DAGs). These graphs are dynamic in nature, as nodes can enter or leave the system at any time and as services can choose to change how they interact.

StreamGen can be used in two modes. First, it can be applied at the `periphery' of a system, where a client built using the StreamGen library generates or injects data streams/requests against service node(s). Request rates can be based on statistical distributions, or they can be derived from realistic workload models like the ones described in [14]. Second, and as shown in Figure 1, StreamGen can also be incorporated within individual service nodes. In this mode, StreamGen enables users to couple outgoing to incoming traffic using its Virtual Connection module (described later). Users can define arbitrarily complex conditions on incoming events. Service code is executed when conditions are satisfied, which may in turn generate outgoing messages.

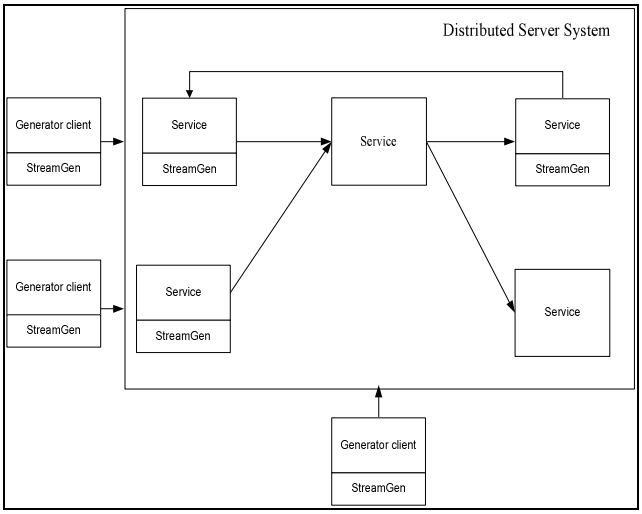

Figure 1 StreamGen in a Distributed System.

2.1. The StreamGen Library

The StreamGen library available for download from our website has the following components: (1) a C library with APIs for workload generation and monitoring, and (2) traces for generating realistic dynamic workloads [14]. The library consists of several modules. The *Scheduler* module is a one-shot timer API. Using the scheduler requires users to incorporate the timer_tick() call in the program's main loop. The *Transport* module provides objects to abstract basic transport concepts such as connections, sessions, and calls. This module also defines a set of basic events that correspond to different transport activities. For example, EV_CONN_CONNECTED is an event that, when fired, indicates that a new connection has been established. Similar events exist for connection closing, failure, message receipt, message transmission, etc. The Transport module implements HTTP, but other transports are added easily. To demonstrate how the different modules can be used, a user can log every outgoing call by simply registering a log_outgoing_call() function to execute whenever an EV_CALL_SEND_STOP event is fired. The *Statistics* module implements data collection and analysis functions. The *Generator* module contains pre-packaged APIs for generating workloads based on statistical distributions, but it is straightforward to program additional generators. Finally, the *Virtual Connection* module provides an implementation of a virtual connection. A virtual connection is built from one or more basic transport connections. Virtual connections can be set up to raise a

Virtual Event when an arbitrary combination of events occurs on the underlying transport connections. A Virtual Connection is an abstraction that monitors N input channels. The user can programmatically define a Virtual Event for this virtual connection that is fired (published) when the associated monitor condition evaluates to TRUE. This enables us to specify and control the information flow not just on the periphery on the system, but at any computational node. For example, we can create a virtual connection that is composed of input channels 1 and 2 (see Figure 2) and define a virtual event to be fired when three input messages have accumulated on input channel 1 and two on the input channel 2. A handler can be registered for this virtual event, which generates an output event on an output when the virtual event is fired.

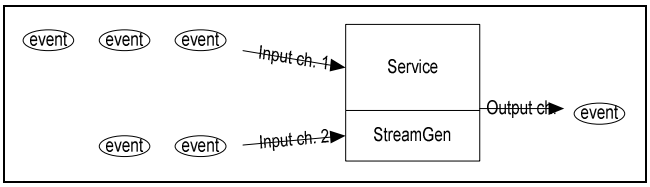

Figure 2: A Virtual Connection Example.

2.2. Library Implementation

The StreamGen library is derived from HttPerf [1], a tool for measuring web server performance. APIs have been added to the HttPerf code for generating workloads derived from FTP traces (explained below), in addition to the Virtual Connection module. HttPerf uses HTTP as the default transport protocol; to use a different transport protocol, new protocol bindings have to be created. This simply means generating (publishing) the appropriate events on the respective protocol events.

Table 1 lists the new APIs for Virtual Connections. Our current implementation limits monitor conditions to simple event counters. In the future, we will add native support for the ECho publish/subscribe event delivery system as the main transport and also extend monitor condition to take advantage of PBIO (ECho's data representation library). With these extensions in place, we will be able to build complex monitor conditions and virtual event handlers that can apply application-specific transformations, personalize and annotate event data, or apply business rules to information streams.

The experiments presented in this paper use ECho as the underlying communication mechanism. In these cases, StreamGen is simply used to inject trace-based workloads into the systems being evaluated. Since we do not use virtual connections or their APIs, StreamGen can be used with a Transport Module into which ECho has not yet been integrated.

Table 1: Virtual Connection APIs.

API	Description
typedef struct VCMonitorCondition { Conn* _connection; Event_Type _ev_type; int _num_events; } VCMonitorCondition;	Typedef for one element of the monitor condition
VCKey* add_virtual_connection(char* expression, VCMonitorCondition *list, int n)	Define a new virtual connection based on data in *list, which* has *n* entries. *expression* is a Boolean expression that we use to combine the conditions in *list*
Void remove_virtual_connection(VCKey* vc)	Cancel a previously defined connection

The following programming actions are required when using StreamGen as a standalone workload generator. A *main*() function generates traffic patterns based on the supplied traces [14], by calling the *ftplog_parse*() API. This API schedules a user-specified callback to be executed for each selected trace, and such traces can be scaled via additional specified arguments. Arguments may be passed programmatically or from command lines. StreamGen also supports statistical workloads (e.g. random, uniform, etc.), by simply using one of the generators supplied with HttpPerf. In these generators, each user callback generates some amount of traffic and then schedules its next execution time based on some statistical distribution.

3. Overview of Sample Applications

This section briefly describes the two classes of applications emulated with StreamGen. The first class is real-time scientific collaboration, represented by generalizing the `SmartPointer' application framework. The second class emulates a server subsystem of an operational information system, termed REDE, derived from the server clusters used by an airline with which we are familiar.

3.1. SmartPointer

SmartPointer is a framework for scientific collaboration first presented in [5]. Output data is

streamed from a large parallel computational science code (molecular dynamics -MD) to a series of annotation and visualization services (see Figure 3). Two facts determine the application's operation. First, current end user needs determine the way in which services are applied to the data produced by the MD code. Second, the resources available to stream data and apply services are subject to dynamic variation. The purpose of using StreamGen, then, is to exercise the framework with different user behaviors, modeled via different request traces, services applied to request traces, and topologies linking such services. Different resource availabilities correspond to different configurations of the underlying hardware testbed.

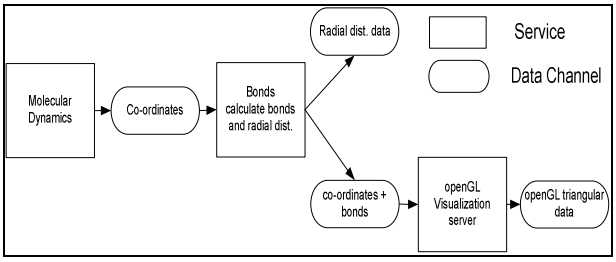

Figure 3: The SmartPointer System.

3.2. Replicated EDE (REDE)

The Replicated Enterprise Data Engine (REDE) system shown in Figure 4 was first described in [13]. The primary server applies business logic to an incoming stream of FAA data events and produces business data events. The system must operate within strict limits on output delays. In addition to FAA data processing, the server also maintains internal state, where new clients connecting to the system request a copy of this state. Since the internal state can be substantial, a large number of initial state requests can impact the server-based delay experienced by other clients. A new mirroring technique addressing this problem is evaluated in Gavrilovska et al. for different levels of constant rates of initial state requests. This paper refines these results, by studying the performance of the REDE cluster under more realistic workloads of initial state requests, generated with StreamGen's dynamic traffic traces. The hope is to gain insights about system performance under transient loads.

4. Using StreamGen for Performance Studies

4.1. Parametric Studies: EDE Enterprise System

To demonstrate the utility of the StreamGen generator, we first turn to the EDE enterprise system. The system's use in previous research [12][13] provides a reasonable initial testing configuration. As in that paper, we study the performance under loads imposed by additional initial state requests. The flexible infrastructure provided by StreamGen, however, permits us to use more realistic transients for the failures and arrivals of external clients. This results in two different experiments, highlighting the ease with which StreamGen can be reconfigured to test applications that stream information under different data assumptions. The desired rate for the "main" output data event is as in our previous work, but we utilize data access patterns for the full initial state requests that are based on the Linux ftp traces provided with StreamGen.

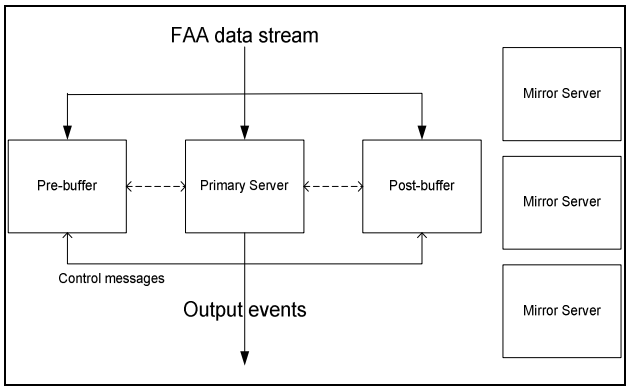

Figure 4: REDE Cluster.

A key parameter in evaluating the ability of a server to efficiently handle imposed load is the ratio of the sizes of the two event types: basic update events and the large, initial state events. The first experiment utilizing the StreamGen implementation for the EDE system uses a basic update event size of 500KB and a large initial state request size of 10MB. The particular traffic pattern chosen for the initial state perturbation is based on the access pattern of the RedHat 7.1 ISO image from the StreamGen library. Figure 5 shows the histogram of this arrival frequency, scaled to an appropriate size. The resulting delay in delivery of the main output data to those clients is shown in Figure 6.

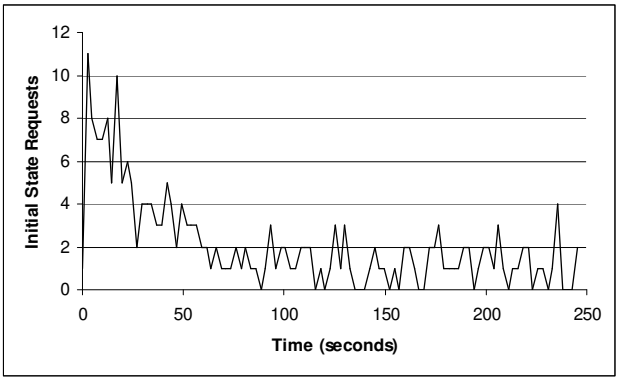

Figure 5 Histogram of Initial State Requests (bin size = 10 seconds)

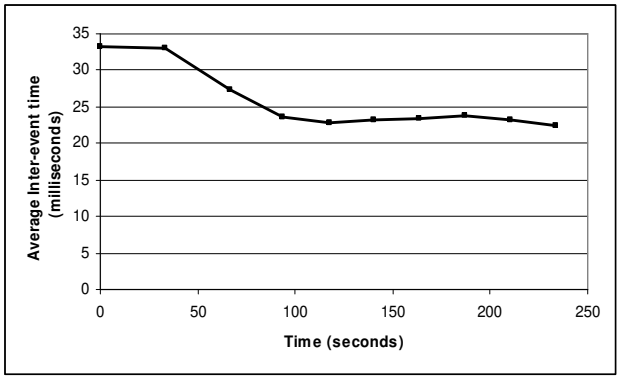

Figure 6 Average Inter-event Delay – each data point averaged over 1000 events.

The experiment was then reconfigured for a run where the main output data size is 500B and the initial state data event is 1MB in size. The hardware is changed to a Sun UltraSparc II platform with a Fast Ethernet (100Mb) interconnect, rather than a Pentium III platform with gigabit Ethernet of the first case. Due to this network bandwidth limitation, the server is able to send 12 initial state events per second. Figure 7 shows the distribution of initial state requests. The resulting delay in delivery of output data to those clients is shown in Figure 8.

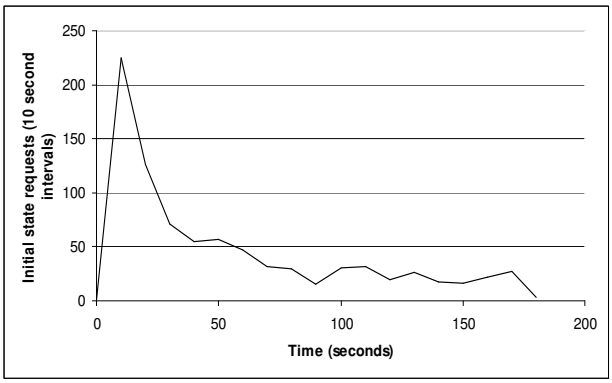

Figure 7 Histogram of Initial State Requests (bin size = 10 sec)

The ease of reconfiguration for these experiments is one advantage of using the StreamGen system: it is straightforward to change client access pattern, including duration, size, and time of processing of each data field (not all of which were fully explored in this example).

This improves the experimenter's ability to explore the potential state space. In these experiments, for example, a qualitative difference exists between the reactions of the same server system to the two different access message size regimes. This can have profound impacts on the design and implementation of a distributed business information service. Another interesting insight from these experiments is derived from comparing them to our previous results reported in [13]. Specifically, we can see that the initial spike in state requests causes clients of data events to experience increased delays of up to 62%, and that this effect lasts well beyond the duration of the spike. Furthermore, the perturbation measured in this paper's experiments is significantly less severe than the one reported in [13] when a less realistic, constant full state request rate was used.

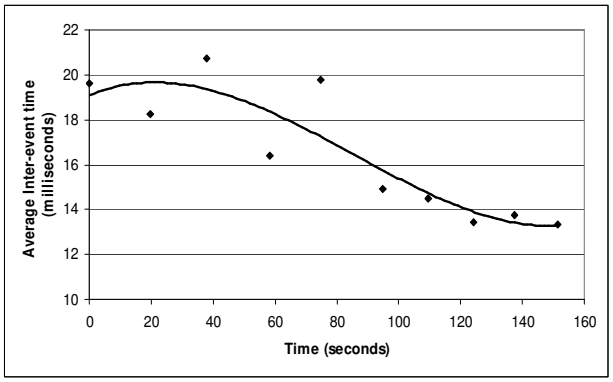

Figure 8 Average Inter-event Delay – each data point averaged over 1000 events.

4.2. Parametric Studies: Scientific Collaboration

A second demonstration of the capabilities of StreamGen uses the SmartPointer [5] distributed visualization and steering system for computational molecular modeling applications. It has a series of component services that annotate, personalize, and transform the scientific data flow. As described above, the system is based on a publish-subscribe semantic to allow transparent data access.

A limitation of the original implementation of SmartPointer is a lack of scalability in the graphics-rendering pipeline. In response, this experiment extends the basic SmartPointer architecture with a cluster of visualization servers and a simple load balancing algorithm to direct new connections to the appropriate server. Using StreamGen we utilize the client request modules to study the resulting system's scalability. The actual software architecture appears in Figure 9. The load-balancing algorithm samples queue lengths and delays at each server. It reacts to server overload by directing new client requests to the channel of the least loaded server. Existing requests are not migrated; they are serviced by the primary server until canceled.

Of particular interest is the following usage scenario: SmartPointer should scale to support wide-area collaborations across multiple researchers. Given the very long-running nature of many computational codes, we expect to see periodic bursts in interest in their output

(i.e., visualization) data. For example, we should see a peak early in the morning as scientists check on the night's results, then similar peaks after lunch and right before leaving for the night. Some scientists will find interesting output results and continue observing and annotating data for long periods of time, while others will wait until sufficient data has accumulated for their needs before viewing it again.

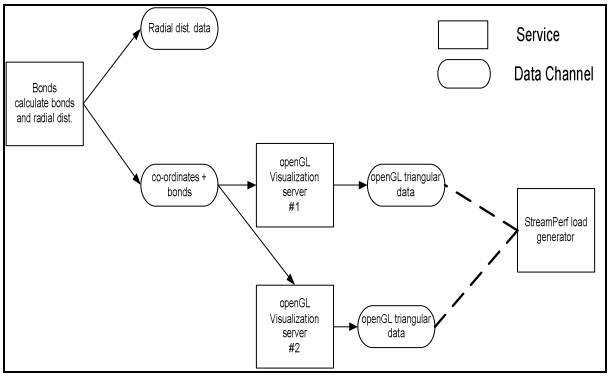

Figure 9 Load Balancing Algorithm Setup

To create a reasonable emulation of such a usage scenario, we utilize an observed traffic pattern from the StreamGen trace library, selecting a trace based on the RedHat 7.3 release data. For this trace, each rpm data download session is mapped to the behavior of some collaborator, and in the StreamGen extension of SmartPointer, it results in the creation of a new channel with a bounded-area specialization filter that represents the client's interests. Filter code represents each participant's choices of personalization and annotation and also associates some computational overhead with each additional stream. Further, each new channel lasts for N number of events, where N is the number of files in the FTP download session. This is analogous to some users watching for short periods of time and others watching for extended periods of time.

From these experiments, it is clear that to handle surges in demand; the openGL servers must be replicated. The question we aim to answer is how many replicates are needed for a certain load.

The experiments conducted use a cluster of seventeen 8-way Pentium III Xeon servers at the Georgia Tech IHPC laboratory. Each of the distributed service processes runs on a separate machine, as well as the client code generator. Figure 10 shows the total number of concurrent connections against the clustered visualization service as a function of time. This reflects both the number of new connections being initiated and the persistence of long-running connections.

As apparent in Figure 11, adaptive load balancing is clearly insufficient in the one- and two-server implementations, but the three-server implementation has sufficient `cycles' to prevent the queue overruns that make the previous two cases untenable. However, note that under the heaviest load, the delay is still considerably larger than desired by the soft real-time constraints the SmartPointer system tries to attain in the single, constant-rate client case.

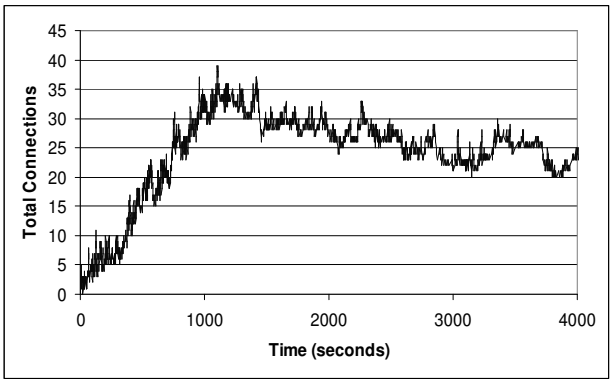

Figure 10 Connections vs.. Visualization Server(s)

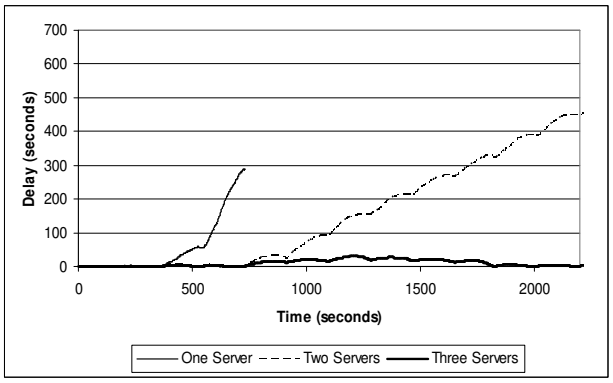

Figure 11 Delays at Different System Configurations

5. Conclusions

This paper describes a workload generation tool, termed StreamGen, for distributed information flow applications. The tool has three major components: a workload generation library derived from the HttPerf tool, support for structured data event types, and traces derived from traffic analyses at the Georgia Tech Linux mirror site. When applying StreamGen to two applications, one from the high-performance scientific domain and the other is an enterprise-scale application, some interesting insights are attained, particularly addressing the behavior of these applications under bursty load conditions. We also show that these insights are more realistic than those derived in

earlier work that did not benefit from StreamGen's dynamic traffic traces.

While our current implementation of the Virtual Connection module still relies on the HTTP transport code provided in the HttPerf tool, StreamGen is easily generalized to other transport methods. This will be demonstrated in our future work by integrating it with our own ECho publish/subscribe communication library and by applying it to the data exchange middleware used by some of our industry partners. In addition, we intend to generalize the condition matching facilities in the current implementation of StreamGen to support arbitrary data transformations, annotations, and stream merging actions, similar to those offered by the Gryphon system [10].

6. References

[1] D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance", WISP, ACM, Madison, WI, June 1998, pp. 59-67

[2] H. Pfneiszl and G. Kotsis, "Benchmarking Parallel Processing Systems - A Survey", Institute of Applied Computer Science and Information Systems, University of Vienna, Technical Report No. TR-96103

[3] M. Seltzer, et. al., "The Case for Application-Specific Benchmarking", In Proceedings of the 1999 Workshop on Hot Topics in Operating Systems

[4] M. Frumkin and R. F. Van der Wijngaart, "NAS Grid Benchmarks: a tool for Grid space exploration", in Proc. 10th Intl. Symp. on High Performance Distributed Computing, August, 2001, pp. 7-9

[5] M. Wolf, Z. Cai, W. Huang and K. Schwan, "SmartPointers: personalized scientific data portals in your hand." In Proc. of the 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland, 2002

[6] G. Eisenhauer, F. Bustamente and K. Schwan, "Event Services for High Performance Computing," In Proc. of High Performance Distributed Computing (HPDC-2000), August 1-4, 2000

[7] Standard Performance Evaluation Corporation, "SPEC HPC2002 V1.0" [online], [cited November 2003], available from World Wide Web: <www.spec.org/hpc2002/>

[8] Transaction Processing Performance Council, "TPC V5" [online], [cited November 2003], available from World Wide Web: <http://www.tpc.org/tpcc/>

[9] Sun Microsystems, "ECPerf" [online], [cited November 2003], available from World Wide Web: <http://java.sun.com/j2ee/ecperf/>

[10] R. Strom, et.al.,. "Gryphon: An information flow based approach to message brokering." In International Symposium on Software Reliability Engineering, 1998

[11] V. Oleson, et. al., "Operational information systems - an example from the airline industry." In First Workshop on Industrial Experiences with Systems Software (WIESS)

[12] A. Gavrilovska, K. Schwan and V. Oleson, "Adaptable Mirroring in Cluster Servers", In Proc. of the 10th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC-10'01), August 2001

[13] A. Gavrilovska, K. Schwan and V. Oleson, "A Practical Approach for Zero' Downtime in an Operational Information System", In Proc. of the 22nd Intl. Conf. on Distributed Computing Systems (ICDCS'02), July 2002

[14] M. Mansour, M. Wolf and K. Schwan, "Dynamic Traffic Behaviors - The GT Mirror Trace," High Performance Grid Computing workshop of IPDPS '04

[15] A. Snavely et. al., "Benchmarks for grid computing: a review of ongoing efforts and future directions", ACM SIGMETRICS Performance Evaluation Review, vol. 30 , no. 4, March 2003

[16] F. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, "Efficient Wire Formats for High Performance Computing," in Proc. of Supercomputing 2000 (SC 2000), Dallas, Texas, November 4-10, 2000