

# **File System Suite of Benchmarks**

**Written: February 2004**

**EP Network Storage Performance Lab**

**Technical Report: TR-2004-001**

©2004 by EP Network Storage Performance Lab, Inc. All Rights Reserved  
Permission is granted for reproduction of the work only in its entirety. This  
copyright notice must be included in the reproduced paper. EP Network Storage  
Performance Lab acknowledges all trademarks herein. The work reported in this  
paper was funded by EP Network Storage Performance Lab.

# Table of Contents

<b>1.0</b>	<b>INTRODUCTION .....</b>	<b>5</b>
1.1	WHY BENCHMARKS?.....	5
1.2	FILE SYSTEM BENCHMARKING.....	5
1.2.1	<i>Existing File System Benchmarks</i> .....	5
1.2.1.1	Andrew File System Benchmark.....	6
1.2.1.2	Bonnie.....	6
1.2.1.3	IOmeter.....	6
1.2.1.4	IOzone.....	6
1.2.1.5	NetBench.....	6
1.2.1.6	PostMark.....	6
1.2.1.7	SPECsfs®.....	7
1.3	NEED FOR A NEW BENCHMARK.....	7
1.3.1	<i>System Level Benchmark</i> .....	7
1.3.2	<i>Component Level Benchmark</i> .....	8
1.3.3	<i>Benchmark Suite</i> .....	8
<b>2.0</b>	<b>REQUIREMENTS FOR FILE SYSTEM BENCHMARK.....</b>	<b>8</b>
2.1	USEFUL.....	8
2.2	FAIR.....	8
2.3	REPRODUCIBILITY.....	9
2.4	WORKLOAD.....	9
2.4.1	<i>Based on Real World</i> .....	9
2.4.2	<i>Extensible</i> .....	10
2.4.3	<i>Operation Clusters</i> .....	10
2.4.4	<i>File System Aging</i> .....	10
2.4.5	<i>File Sharing</i> .....	11
2.5	PLATFORMS SUPPORTED.....	11
2.6	SCALABILITY.....	11
2.7	SYSTEM LEVEL BENCHMARK SUPPORT.....	12
2.8	COMPONENT LEVEL BENCHMARK SUPPORT.....	12
2.9	SECURITY.....	12
2.10	METRICS.....	12
2.10.1	<i>Response Time</i> .....	13
2.10.2	<i>Throughput</i> .....	13
2.10.3	<i>Resource Utilization</i> .....	13
2.10.4	<i>Efficiency</i> .....	13
<b>3.0</b>	<b>FILE SYSTEM BENCHMARK ARCHITECTURE .....</b>	<b>13</b>
3.1	BENCHMARK COMPONENTS.....	14
3.2	BENCHMARK MASTER.....	14
3.3	BENCHMARK DESCRIPTION FILE.....	15
3.3.1	<i>Benchmark Specific Parameters</i> .....	15
3.3.2	<i>Workload Specific Parameters</i> .....	17
3.3.3	<i>General Parameters</i> .....	17
3.4	WORKLOAD DESCRIPTION FILE.....	18
3.4.1	<i>Distribution Methods</i> .....	18
3.4.2	<i>File Types</i> .....	19
3.4.3	<i>File Set Related Parameters</i> .....	19
3.4.4	<i>File Request Related Parameters</i> .....	20
3.5	LOAD GENERATOR.....	21
3.5.1	<i>Threads</i> .....	22
3.5.1.1	<i>Main Thread</i> .....	22
3.5.1.2	<i>Communication Threads</i> .....	22

- 3.5.1.3 Op Generator Thread..... 22
- 3.5.2 Workload Mapping ..... 23
- 3.5.3 File Request Clustering ..... 23
- 3.5.4 File Sharing ..... 24
- 3.6 BENCHMARKING PROTOCOL..... 25
- 3.7 BENCHMARK PHASES ..... 25
  - 3.7.1 Load Generator Setup..... 26
  - 3.7.2 Idle..... 26
  - 3.7.3 Populate File System ..... 26
  - 3.7.4 Warm Up..... 27
  - 3.7.5 Measurement..... 27
  - 3.7.6 Result Collection..... 27
- 3.8 BENCHMARK INSTALLATION ..... 27
- 4.0 WORKLOAD..... 27**
- 5.0 TERMINOLOGY ..... 28**
- 6.0 REFERENCES ..... 29**

## List of Figures

Figure 1: Benchmark Components .....	14
Figure 2: Directory File Sharing.....	24
Figure 3: Load Generator Phases.....	26

## List of Tables

Table 3.1: Type of Generated Operation .....	23
--	----

## 1.0 Introduction

This paper discusses a new suite of file system benchmarks called the *File System Suite of Benchmarks* (FSSB). The implementation of the FSSB is currently being worked on at EP Network Storage Performance Lab.

### 1.1 Why Benchmarks?

It would be nice if a performance engineer could take an application at an end-user site and have real users use it and measure meaningful performance. If a performance engineer tried to do this they would quickly learn that the results vary from day to day, hour to hour, and even minute to minute. The variability is due to a constantly changing environment which is the result of the end-users doing different things at different times, or other users running different applications competing for computing resources that the users being measured were using.

Benchmarks were created to define a baseline from which to measure performance against. A good benchmark should output a performance measurement **A** for a given hardware and software configuration. When no changes are made to the hardware and software then a good benchmark should output the same performance measurement **A** within some small margin of error. A change could then be made to the configuration and the benchmark run again and output a performance measurement **B**. The difference between **A** and **B** would indicate the impact the change had on the measured performance. If **A** and **B** were equal, within some small margin of error, then that would indicate that the configuration change was not a performance factor for the benchmark.

Computer benchmarking is used in engineering organizations to determine the best approach to implementing a solution, by test organizations to test functionality and to look for performance regressions, by marketing organizations to generate data to show that their system is better than everybody else in the market, and by end-users to determine if a computing solution will provide the performance that their business operations require. Performance data is also important for showing that a vendor's solution will scale with business growth.

### 1.2 File System Benchmarking

File system benchmarking is one particular aspect of computer benchmarking that has seen a recent increase in activity. In the late 80s and early 90s there were several benchmarks developed, each focused on a particular aspect of the file system. Some benchmarks were file system independent while others were specific to a particular implementation. If the benchmark came out of a corporation, the typical goal of the benchmark was to show that the company's new feature was better than everybody else's. Benchmarks that came out of the research community or performance organizations tended to be more vendor neutral. Examples of current research in this area can be found at Harvard College and Stony Brook University [Aran04] [Smit01] [Tang95].

#### 1.2.1 Existing File System Benchmarks

This section will present a quick overview of existing file system benchmarks.

### **1.2.1.1 Andrew File System Benchmark**

The Andrew Benchmark came out of the Andrew File System (AFS) work at Carnegie Mellon University. This benchmark tries to capture a typical file server workload in a software development environment. It will support N number of clients each generating load against the server. The benchmark is protocol independent allowing comparisons with NFS. Although this benchmark was very popular in the early 90s it is rarely used outside of the research community today.

### **1.2.1.2 Bonnie**

Bonnie is a simple tool that tests the throughput of a file system but the results do not map to any real-world workload. Bonnie uses standard C library calls, which makes it portable, but it only measures the file system on the computer that it is run on and does not scale to multiple computers. Bonnie can be obtained over the Internet.

### **1.2.1.3 IOmeter**

IOmeter is currently distributed as an open-source project. The benchmark was originally developed by Intel and is good for generating I/O workloads and collecting the resulting response time, throughput, and CPU usage. IO generators are typically good for stressing file systems or testing block devices directly. They do not do well as general purpose file system benchmarks because they do not exercise all the file system operations.

### **1.2.1.4 IOzone**

IOzone is a full featured, open-source file system benchmark tool that measures I/O throughput. It can generate a wide variety of access patterns. This benchmark has been used extensively as a tool for diagnosing and debugging performance issues with SAN and NAS storage. IOzone can be obtained over the Internet.

### **1.2.1.5 NetBench**

NetBench is a benchmark for Common Internet File System (CIFS) file servers. It will support N number of clients each generating load against the server. The benchmark is protocol specific. The benchmark defines a mix of operations that it generates and the workload is extremely light weight with some results reporting 40 clients running off of 4 disk drives. NetBench binaries are freely available from the Internet but the source code is controlled by Veritest, a division of Lionbridge Technologies, Inc., which purchased Ziff-Davis, Inc.

### **1.2.1.6 PostMark**

PostMark is a benchmark that is used for testing small file environments such as e-mail and netnews servers. PostMark will run on local or remote file systems. PostMark was created by Network Appliance, Inc. PostMark can be obtained from the Internet.

### **1.2.1.7 SPECsfs®**

The Standard Performance Evaluation Corporation (SPEC) System File Server (SFS) benchmark measures Network File System (NFS) server performance [SPEC01]. It will support N number of clients each generating load against the server. The benchmark is protocol specific. The benchmark defines a mix of operations that it generates and outputs data for a throughput versus response time graph. The benchmark can be purchased from SPEC.

## **1.3 Need for a New Benchmark**

There has been a substantial change in file system technology that warrants the need for a new file system benchmark. The first major change is that there are now fundamentally two different types of Operating Systems – Microsoft Windows® and UNIX®. The UNIX operating system has successfully moved into the high end while Windows dominates the low end. That is all changing as Windows is trying to push into the high end and Linux is pushing UNIX into the low end space. Between these two operating systems are over 10 different file system types and 2 different network-based file systems. Users would like to be able to compare the performance between a Window-based file system and a UNIX-based file system, not only locally but over the network. They want to know if their application will run better on Windows against a UNIX-based file server or will if it runs better against a Windows-based file server.

The second major change is that file servers have taken on a new roll as part of the network storage solution space. Even the name has changed from file servers to Network Attached Storage (NAS). The days of using general purpose computers as file servers are all but gone. File servers today are specialized storage products. There is a big push to make N storage boxes appear as one file system to the customer. This is usually accomplished with some type of global file system.

The third major change is the introduction of iSCSI which allows the SCSI disk access protocol to be run over Ethernet. The protocol accesses network storage at the block level while NAS file server protocols access network storage at the file level. Accessing network storage at the file level usually makes storage administration easier but the downside is that file level accesses tend to be slower than block level accesses. End-users want to be able to compare file level access performance to that of block level.

Existing benchmarks are ill equipped to deal with the change going on in the file system space. A new benchmark is needed that can measure these new features and allow end users to make intelligent choices when it comes to comparing computing solutions. Existing file system benchmarks can be broken down into two categories which are covered below.

### **1.3.1 System Level Benchmark**

A system level benchmark runs directly on the computers being measured. It can consist of real-world applications or it can use synthetic workloads derived from analyzing the behavior of real-world applications. System level benchmarks are protocol-independent allowing the comparison of different network file systems, and can be scalable allowing the benchmarking of complex hardware configurations.

### 1.3.2 Component Level Benchmark

A component level benchmark measures the performance of a component of the entire system. Examples are: Local File System, NFS Client, or a CIFS Client. Component level benchmarks are also available for measuring file server performance. Examples are: NFS Server, or a CIFS Server. Component level benchmarks tend to be protocol dependent, and can be scalable allowing the benchmarking of complex hardware configurations.

### 1.3.3 Benchmark Suite

Both system level and component level benchmarks are needed. End-users want system level benchmarks to use to determine which solution to their problem is better. Developers typically want component level benchmarks. They want a tool that measures an isolated area of the system so they can work on improving the performance. There is a mixture of end-users, Value Added Resellers (VAR), and developers that would use both types of benchmarks.

It would be nice to have a benchmarking suite that included both system level and component level benchmarks. This would provide a single tool with the same interface for everybody to use. It would be important to have some type of workload mapping function so that results from benchmarking an NFS server could be compared to a CIFS server.

The rest of this paper discusses the requirements and architecture for the FSSB.

## 2.0 Requirements for File System Benchmark

This section goes over the high level requirements of the FSSB.

### 2.1 *Useful*

A benchmark needs to generate meaningful results. It is unrealistic to expect the results of a single benchmark run to map to all cases in the real world. Even so, a benchmark should produce reproducible results that allow different systems to be compared in such a way as to determine which systems handle the workload the best. Note that a system could be computer hardware or different software components running on the same computer hardware.

### 2.2 *Fair*

FSSB should be a fair measure of file system functionality and performance. It is interesting to note that several existing file system benchmarks were created to show off a new feature. For example, one of the first NFS file server benchmarks was developed by Legato to show that their Prestoserve® product would significantly improve the performance of synchronous writes on the NFS server. A problem arose as more and more companies started using this benchmark to measure NFS server performance. The problem was that one company would tweak the benchmark or its workload to make their performance results look better and then publish those results. Customers would try to compare the results from two now different workloads to determine which system was better. This wasn't a fair comparison. After a while the NFS server companies decided to declare a truce and form the LADDIS consortium which became the SPEC SFS subcommittee. The subcommittee allows competing computer equipment vendors to get

together to agree on the benchmark and workload that they all will use to compare file server performance results.

### **2.3 Reproducibility**

A key component of any benchmark is reproducibility, that is for a given hardware setup the same results are reported over and over again. Of course, that assumes that the hardware that the benchmark is being run on is dedicated for the benchmark and that nobody else is using it. Even with a good benchmark it is sometimes difficult to get reproducible results due to several factors. One factor could be scheduled background jobs being run on one or more of the computer systems. One trick performance engineers use to work around this problem is to set the system time to a fixed value before every run. Another factor that can impact the reproducibility of benchmark results is extraneous traffic on the computer network that the performance engineer was not aware of. For example, network interfaces on the SUT may still be connected to the corporate network.

Other factors have to do with the state of the system. A file system in effect stores state about the system. A performance engineer may run a benchmark on a file system 10 times and get 10 different results. This does not necessarily mean that the benchmark is bad but could indicate that the state in the file system needs to be reset before the benchmark is run. Typically file systems are created new before each run to ensure that the benchmark results are reproducible. Even after doing this the measured performance results still may not be consistent. This can be due to the operating system keeping state in terms of file data caches and operating system caches. Cached data for a file system should be flushed when a file system is un-mounted and remade. Even so, the operating system may have other data caches for storage volumes or created pools of threads that are now waiting for work. If there is still variability then the performance engineer should restart the system before running the benchmark to minimize the variability.

### **2.4 Workload**

Determining the workload for a benchmark is perhaps the most difficult task in creating a benchmark. The workload could be a single application or a simulation of multiple applications running at the same time. The workload could be derived from measurements of file system operations running in real world environments. For example, the work load could be derived from the load generated from reading mail, editing documents, working on engineering designs, compiling, and other real world activities. The section FSSB workload requirements are listed below.

#### **2.4.1 Based on Real World**

The workload should represent activity found in the real world which can be determined if the workload matches the real application in the following three areas [Jain91]:

1. **Arrival Rate:** The arrival rate of requests should be the same or proportional to that of the application.

2. Resource Demands: The total demands on each of the key resources should be the same or proportional to that of the application.
3. Resource Usage Profile: Resource usage profile relates to the sequence of the amounts in which different resources are used in an application.

The workload needs to represent a modern real world workload. This is difficult since collecting application file access traces is time consuming and there are not many publicly available. The ones that are available are usually several years old.

#### 2.4.2 Extensible

One lesson learned from previous file system benchmarks is that one single workload does not satisfy the needs of the user community. FSSB should not be tied to one single workload. It should be extensible and handle several different workloads, it should allow workloads to be added, and it should be easy to add new features to in the future.

#### 2.4.3 Operation Clusters

When looking at file system operations there are obvious dependencies between the operations. For example, a file cannot be read unless it has first been opened. An operation cluster would be a set of operations that need to execute in a specific order. An example would be an open followed by one or more reads and then the close operation. FSSB should support the specification of operation clusters in the workload.

Some operation clusters are not so obvious. For example, for the [Rose00] workload it was noted that files less than 20KB are typically read in their entirety that is sequentially from the beginning to the end. That could imply the following operation cluster: *open*, an *fstat* to get the size of the file so the size of buffer to allocate is known, a *malloc* of a buffer, a *read* of the entire file into the buffer, and a *close*. Due to these operation dependencies FSSB should support the option of having operations like *malloc* in the operation mix even though they are not a file system operation.

#### 2.4.4 File System Aging

The issues of file system aging and the effects on file system performance have been raised by [Smit97]. In the benchmarking world, file system aging is a difficult beast to address. A benchmark can be configured to run the warm up phase for minutes, hours, days, weeks, or months to get the “right” age for the file system whatever that may be. The problem is aging the file system can take a significant amount of time to run which is not a practical requirement. Using backup and restore software doesn’t address the problem because they do not preserve the on-medium format of the file system. Another approach is to take a pre-aged file system image and load it onto the SUT and benchmark it. While this approach appears to solve the problem, it raises new problems. The on-medium format of a file system can change from one OS release to another and can even be modified by an OS patch. Incompatibles between on-medium file system formats can lead to catastrophic failures on the SUT. Using pre-aged file system images is a good solution for a developer studying the performance of an aged file system since they can

deal with the complexities of using file system images. It is not a reasonable approach for the average performance engineer or end user.

The only support for file system aging in FSSB is through the warm up phase.

#### 2.4.5 File Sharing

The sharing of files between different users and/or applications occurs on a regular basis. For example, mail reading can involve the sharing of the inbox file between the thread that displays the mail to the user and the thread that updates the inbox file with the latest mail. The extent of file sharing can vary between operating systems and network file system implementations. Applications tend to share directories and sometimes even files. Executables and libraries are examples of files that are read by many network file system clients but never modified by the client.

There are two locking paradigms in use by UNIX and Windows systems. The first is *mandatory locking* where applications are forced to obey the restrictions imposed by a file lock. The goal is to make it impossible for an application to violate a file lock. Windows systems implement mandatory locking. The second is *advisory locking* where the application can choose to obey the file lock. The downside to advisory locking is that an application that is unaware of file locking can access or update locked files. UNIX systems typically implement advisory locking, although on some UNIX implementations mandatory locking for a file can be enabled by changing the file mode. The two different locking paradigms have had a curious effect on the underlying systems. UNIX has historically had issues with file sharing until the middle of the 1990s simply because file locking was not heavily used on UNIX. Windows on the other hand forced developers of applications to deal with the issue of file locking early. One thought on this is that UNIX has taught its users not to share files while the Windows environment encourages it. If true, then a workload study should prove this. Unfortunately the workload studies that were evaluated did not appear to capture file locking operations [Rose00] [Voge99].

FSSB should support read-only sharing of files to model access to executables and libraries, and it should support file locking to simulate the sharing of a common file by applications and or users. The issue of mandatory versus advisory locking is not a problem here since FSSB will generate file locking requests where desired and will know which files it is using file locking on. In effect, FSSB will enforce mandatory file locking.

### 2.5 Platforms Supported

A file system benchmark should be written such that it is portable to the majority of operating systems. This is important for widespread use of FSSB.

### 2.6 Scalability

A file system benchmark should scale from benchmarking a single system in a simple configuration to benchmarking multiple systems in a complex configuration. FSSB should not care how the file system is configured and should be able to handle systems with virtualized file systems that have any number of systems providing services.

Workload scaling is an important issue. The size of the total file working set and active file working set are chosen so that the SUT cannot possibly cache the entire results, or that the amount of hardware required to do so would be prohibitive. This issue was addressed in the SPECsfs® benchmark by creating a parameter that tied the size of the file working sets to the number of operations requested. The logic in doing this is that a smaller system would peak out at a lower number of NFS operations and therefore typically have a smaller configuration. A larger system would have a higher number of NFS operations and would require a larger configuration to satisfy the requests. At this time a better approach is not evident and therefore this benchmark has a similar parameter. Modern virtualized storage should be monitored to make sure it does not easily create configurations that work around the intent of a workload to have FSSB measure the performance of the storage solution including I/O to cache and the storage medium, but not the performance of I/O solely to cache memory. A workload could be defined to have the intent of measuring I/O solely to cache memory.

## **2.7 System Level Benchmark Support**

A system level benchmark should be independent of any protocol used to implement the file system. A protocol independent benchmark allows the performance comparison of a network file system to a local file system, or the performance comparison of different network file system protocols. For example, NFS and CIFS performance could be compared. FSSB should support system level benchmarking.

## **2.8 Component Level Benchmark Support**

A component level benchmark measures the performance of one component of the system level solution. For example that could be the performance of an NFS Client, CIFS Client, NFS Server, or a CIFS Server. The examples given imply a protocol specific benchmark with the downside being the need to create a benchmark for each protocol, in other words there would be an NFS server benchmark and a CIFS server benchmark. Component level benchmarks should scale just like system level benchmarks although it is important that they work well with minimal setups so developers can use them. FSSB should support component level benchmarking.

## **2.9 Security**

FSSB should allow security parameters to be part of the workload specification. This will give the end-users the ability to determine the performance impact of one security approach over the other. It will also provide a tool so that developers can focus on reducing the performance impact of their security solutions.

## **2.10 Metrics**

Any benchmark requires a set of performance criteria or metrics that must be chosen. The metrics are reported by FSSB and allow one run of FSSB to be compared to another. FSSB will use the metrics listed below.

### 2.10.1 Response Time

The metric response time is defined as the amount of time that elapsed between the file system operation request and the reply. The response time is only calculated for a request that successfully complete, that is the operation returned the expected reply. The response time of operations that do not successfully complete or that timeout are not processed. Instead these operations are marked as having failed.

### 2.10.2 Throughput

The metric throughput is defined as the rate at which the requests can be serviced by the file system. The rate is reported as operations per second (OPs).

### 2.10.3 Resource Utilization

The utilization of a resource is measured as the fraction of time the resource is busy servicing file system requests. Initially, the CPU is the only resource that will be measured by FSSB. This will allow end-users to determine how much CPU a particular solution will consume for a given workload. This metric will be evolved over time.

### 2.10.4 Efficiency

Efficiency is a metric that may be of interest to end-users who are comparing file system workloads on different computer systems. For this benchmark, the term efficiency is the ratio of a measured generic file system value to the measured actual file system value. FSSB will report efficiency numbers for the following:

- File system interface data divided by the underlying I/O data (network or storage medium).
- File system operations divided by network operations.

This metric will be evolved over time.

## 3.0 File System Benchmark Architecture

The goal was to design a file system benchmark architecture that satisfies the requirements of the previous section. One of the big issues was how to resolve the difference between system and component level benchmarking. It doesn't make sense to implement independent system level and component level benchmarks for significant pieces of a file system especially if there is no correlation between the results from the different benchmarks. It would be nice if a benchmarking tool existed that could do both the system level and component level benchmarking using the same workloads. That is users using component level benchmarks would be using similar workloads as those used by the complete file system benchmark.

A file system benchmark suite was decided on that included system and component level benchmarks. The rest of this section describes the architecture of FSSB. It is expected that this architecture will evolve over time and that there may be differences between the architecture

described here and the implementation of FSSB. Note that in the following discussion 1 KB = 1,000 bytes, 1 MB = 1,000,000 bytes, and 1 GB = 1,000,000,000 bytes.

### 3.1 Benchmark Components

The two major components of FSSB are the benchmark master and the load generator. The components are shown in Figure 1 and are described in the following sections. Note that there is only one benchmark master but L number of load generators. There can be only one load generator process on a single computer. There is no limit to the number of computer systems that can take part in the benchmark. The restriction of one load generator process per computer may be relaxed in the future to allow using more than one workload during a benchmark run. For example, a software development workload and an office user workload could be running concurrently. The rest of this section will cover FSSB components.

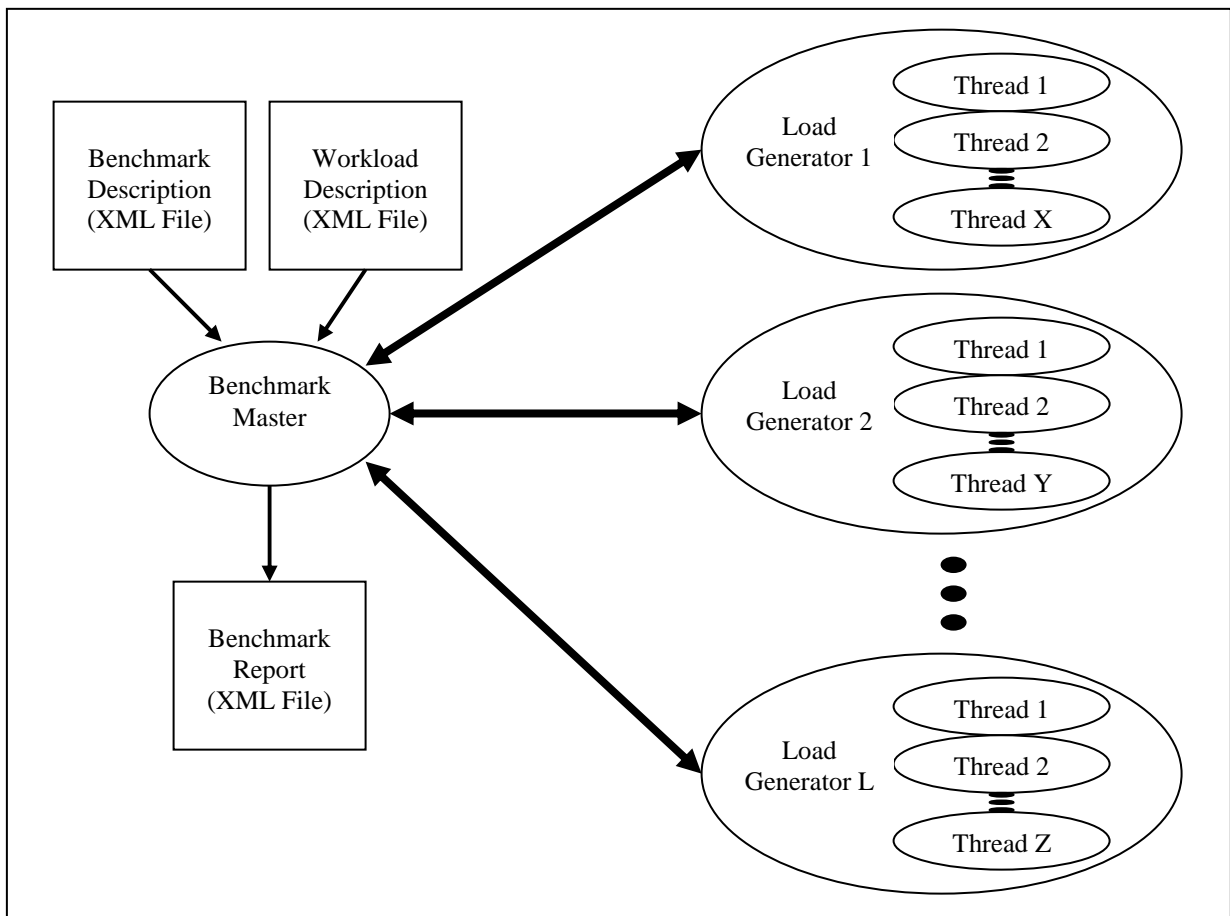


Figure 1: Benchmark Components

### 3.2 Benchmark Master

There is only one benchmark master and it runs on the computer system where the user started FSSB. There can be a load generator running on the same computer as the benchmark master but

that is not a requirement. In other words, the benchmark master can run by itself on a computer system and communicate with other load generators running on other computer systems.

The benchmark master performs the following tasks:

- Parses the benchmark description file and sets the benchmarking variables accordingly.
- Spawns the load generators on the computer systems.
- Reliably steps the load generators through all phases of the benchmarking process.
- Communicates with the load generators using a custom network-based benchmarking protocol.
- Generates the benchmark report.

Every command issued by an instance of the benchmark master has a unique command number which allows all command results from the load generator to be tied back to the command request.

### **3.3 Benchmark Description File**

The benchmark description file describes what the user would like the benchmark to do. The list of parameters will be expanded during the implementation of FSSB. The parameters that can be defined by the user are:

#### **3.3.1 Benchmark Specific Parameters**

##### **BENCH\_TYPE**

This parameter specifies whether the benchmark run is for a component or system.

##### **BENCH\_COMPONENT**

This parameter is only processed if the BENCH\_TYPE is component. It specifies which component is being benchmarked. The currently supported component types are NFS Client, NFS Server, CIFS client, and CIFS server.

##### **BENCH\_RUN\_TIME**

Specifies in seconds how long each measurement phase should last.

##### **BENCH\_WARM\_UP**

Specifies in seconds how long each warm up phase should last.

## BENCH\_DISCOVERY

This parameter is set to YES or NO. If set to YES then the benchmark will determine the optimum threads per load generator and automatically discover the knee of the curve. When this option is set the following parameters are ignored:

WORKLOAD\_REQUESTED\_LOAD

WORKLOAD\_REQUESTED\_LOAD\_INC

WORKLOAD\_REQUESTED\_NUM\_RUNS

WORKLOAD\_THREADS\_PER\_LG

The following parameters are specific to the specified BENCH\_COMPONENT.

### NFS Client or Server Benchmark Component

BENCH\_NFS\_VERSION

This parameter specifies the version number of the NFS protocol to use. The default is to use the newest version of the protocol available.

BENCH\_NFS\_NET\_PROTOCOL

This parameter specifies the network protocol to use. Valid values are TCP or UDP. The default value is TCP.

BENCH\_NFS\_BIOD\_MAX\_READS

This parameter specifies the maximum number of read requests to be outstanding from a single request generating thread. The default value is 2. This parameter is designed to simulate NFS client block I/O daemon (biod) behavior.

BENCH\_NFS\_BIOD\_MAX\_WRITES

This parameter specifies the maximum number of write requests to be outstanding from a single request generating thread. The default value is 2. This parameter is designed to simulate NFS client block I/O daemon (biod) behavior.

### CIFS Client Benchmark Component

These parameters are to be determined.

## CIFS Server Benchmark Component

These parameters are to be determined.

### 3.3.2 Workload Specific Parameters

#### WORKLOAD\_TYPE

This parameter specifies the type of the workload which is converted into a filename that contains the description of the workload.

#### WORKLOAD\_REQUESTED\_LOAD

This parameter specifies the starting number of requests to be generated per second during the warm up and measurement phases.

#### WORKLOAD\_REQUESTED\_LOAD\_INC

This parameter specifies the number to add to the current requested load value before starting the next benchmark run. This parameter is used when more than one benchmark run will be performed for this workload.

#### WORKLOAD\_REQUESTED\_NUM\_RUNS

This parameter specifies the number of benchmark runs to make. The `WORKLOAD_REQUESTED_LOAD_INC` is added to the current requested load value to get a new requested load value before starting the next benchmark run.

#### WORKLOAD\_SYSTEMS

This parameter contains a string of system names, separated by spaces, which will have a load generator process started on them and running this workload.

#### WORKLOAD\_TARGET\_FS

The parameter contains a string of file system names, separated by spaces, which will have load put on them by the load generating threads.

#### WORKLOAD\_THREADS\_PER\_LG

This parameter specifies the number of threads to be spawned to generate load for this given workload.

### 3.3.3 General Parameters

#### GEN\_MON\_SCRIPTS

This parameter specifies the name of a script that is invoked before and after the measurement phase. The script is called with the argument `START` right before

the measurement phase starts and it is called with the argument END right after the measurement phase ends. The use of monitoring scripts can skew the measured results and this parameter should not be set during runs where the results will be published.

### **3.4 Workload Description File**

Each synthetic workload is described in a workload description file. A file is used instead of hard coding the data into the benchmark to allow workloads to be added to the benchmark. The distribution methods available, the file types, and the parameters that can be set in a workload description file are covered in the following sections.

#### **3.4.1 Distribution Methods**

The benchmark supports specifying probability distributions that are used for generating nonuniform values like size of files or the number of files in a directory. This is accomplished by using the inverse transformation of the Cumulative Distribution Function (CDF) [Jain91]. The benchmark can handle continuous or discrete inverse CDFs. Support for distributions can easily be added to the benchmark which currently supports the following distribution methods:

##### **PARETO**

The Pareto Distribution is a power curve that can be easily fit to observed data.

##### **POISSON**

The Poisson Distribution is a limiting form of the binomial distribution and is used extensively in queuing models to specify the number of arrivals over a given interval. This distribution works best when the arrivals are from a large number of independent sources.

##### **UNIFORM**

The Uniform Distribution is used when a random variable is bounded and no further information is available.

Each distribution method takes the following parameters:

##### **Name**

This parameter specifies the name of the distribution to be used.

##### **Type**

This parameter specifies whether the distribution is continuous or discrete. The library will use an internal function to generate values if the distribution is continuous.

### Minimum Value

This parameter specifies the minimum value that can be returned.

### Maximum Value

This parameter specifies the maximum value that can be returned.

The following parameters are only specified if the distribution type is discrete.

### Number of Entries

This parameter specifies the number of entries that are included in the discrete inverse CDF.

The rest of the inverse CDF is specified by a list of inequalities and their associated probability.

### Left Side of Inequality

This parameter specifies the number of entries that are included in the discrete inverse CDF.

### Right Side of Inequality

This parameter specifies the number of entries that are included in the discrete inverse CDF.

### Probability Associated with Inequality

This parameter specifies the number of entries that are included in the discrete inverse CDF.

## 3.4.2 File Types

FSSB supports the following file types:

REG A regular file.

DIR A directory file.

LNK A link file.

## 3.4.3 File Set Related Parameters

FSSB supports the following file set parameters:

#### FILE\_SET\_MB\_PER\_OP

This parameter specifies the number MBs of files that must be created on the file system for every file system operation requested. For example, if this parameter was set to 10 MB per operation then if the user requested 1,000 operations then the file set size would be 10,000 MBs.

#### FILE\_SET\_TYPE\_MIX\_REG

This parameter specifies the percentage of files that are of type REG.

#### FILE\_SET\_TYPE\_MIX\_DIR

This parameter specifies the percentage of files that are of type DIR.

#### FILE\_SET\_TYPE\_MIX\_LNK

This parameter specifies the percentage of files that are of type LNK.

#### File Name Distribution List

This parameter list specifies how file names are generated.

#### File Set Size Distribution List

This parameter list specifies the distribution of file sizes.

#### File Set Size Directory Distribution List

This parameter list specifies the distribution of the number of files inside a directory.

### 3.4.4 File Request Related Parameters

FSSB supports the following file request parameters:

#### File Request Mix List

This parameter specifies the percent that each operation contributes to the total operation mix and the percent of these operations that should be generated from a cluster operation.

#### FILE\_REQ\_SHARED\_RW\_FILES

This parameter specifies the percent of files that are shared between load generators. These files are both read and written by the load generators. This implies that accesses to these files are protected by file locking mechanisms.

#### FILE\_REQ\_SHARED\_RO\_FILES

This parameter specifies the percent of files that are shared between load generators. These files are only read by the load generators and no locking mechanism is used. This parameter models executables and shared libraries that are typically shared among many users.

#### FILE\_REQ\_SHARED\_DIRS

This parameter specifies the percent of directories that are shared between load generators. A shared directory is one in which two or more load generators will use to store files.

#### FILE\_REQ\_PER\_FILE\_OVERWRITES

This parameter specifies the percent of files that are overwritten.

#### FILE\_REQ\_EACH\_FILE\_OVERWRITES

This parameter specifies the number of times a file is overwritten.

#### FILE\_REQ\_READ\_WHOLE\_FILE\_MAX\_SIZE

This parameter specifies that any read operations to files of this size, in KBs, or smaller should be read in their entirety. That means the file should be read starting at offset 0 with a single read command.

#### FILE\_REQ\_ACCESS\_RANDOM\_FILE\_MIN\_SIZE

This parameter specifies that any read or write operations to files at least this size, in KBs, or larger should be accessed randomly. That means that the file operation (read/write) selected should always be preceded with an lseek call.

#### FILE\_REQ\_DYNAMIC\_MEM\_ALLOC

This parameter specifies whether or not to do memory allocations for buffers on the fly. If set to YES then memory is allocated and freed on fly. If set to NO then buffers are pre-allocated before the benchmark starts. This parameter is useful for simulating memory allocations.

#### FILE\_REQ\_ACCESS\_PATTERN

This parameter specifies the file access pattern which is a function of the file size.

### **3.5 Load Generator**

The load generator is the program that generates the load on the component or system being benchmarked. The load generator consists of many threads that perform the following tasks:

- Handles receiving and sending requests from the benchmark master.
- Runs through the different benchmarking phases per the benchmark master instructions.
- Generates the operations to the component being benchmarked ensuring that the requested load is achieved if possible.

The rest of this section will cover the functions performed by the load generator.

### 3.5.1 Threads

Threads are needed because when an I/O operation is generated the thread of execution may be put to sleep to wait for the completion of the I/O operation. The benchmark can issue more than one I/O at a time by having more than one thread of execution. The second reason for having threads is that the computer running the threads may have the ability to support more than one concurrent thread of execution (ex: multiprocessor, hyper-threading, ...). Therefore, it is very important that the threads be somewhat independent of each other and allowed to run freely.

Multi-threading in a benchmark can be tricky and care should be taken to eliminate the use of locks where possible and make sure that the locking hierarchy implemented does not introduce artificial sequence points. The FSSB benchmark takes a minimalist approach to locking.

#### 3.5.1.1 Main Thread

The main load generator thread is the thread of execution that exists when the load generator program is run. It creates all of the other threads and then runs through the different benchmarking phases based on input from the benchmark master. It will handle communicating information from the load generator back to the benchmark master. This thread will never go to sleep waiting for an event other than going to sleep for some amount of time if it has no work to perform.

#### 3.5.1.2 Communication Threads

The load generator creates a pair of threads to handle receiving and sending requests from the benchmark master. The receiving thread sleeps on a network connection waiting for communications from the benchmark master. When a command comes in the receiving thread will place it on the queue for the main thread. The transmitting thread sleeps on the queue from the main thread where it waits for commands to be placed in the queue. When that happens it wakes up and transmits the information to the benchmark master.

#### 3.5.1.3 Op Generator Thread

The load generator creates some number of Op Generator threads that the user requested in the benchmark description file. The purpose of the operation generating threads is to simulate access to the file system.

Op generating threads are where the benchmark specific operations are issued to the component or system being benchmarked. There are four types of file systems that will be initially supported. They are the POSIX File System, NT File System, NFS Server, and CIFS Server. Table 3.1 shows the type of file system operation generated given the values for BENCH\_TYPE and BENCH\_COMPONENT. The operation threads are bound to an array of operation method addresses during the load generator initialization phase. The type of operation methods used is determined from the BENCH\_TYPE and BENCH\_COMPONENT variables specified in the benchmark description file. The load generator has a heuristic to determine if it should use the POSIX or NTFS load generating thread.

BENCH_TYPE	BENCH_COMPONENT	Type of Operation Generated
System	Ignored	POSIX or NT File System
Component	NFS Client	POSIX or NT File System
Component	NFS Server	NFS Server
Component	CIFS Client	POSIX or NT File System
Component	CIFS Server	CIFS Server

**Table 3.1: Type of Generated Operation**

### 3.5.2 Workload Mapping

Workloads are specified using generic file operations that are operating system independent but the benchmarking of a file system is operating specific and the component level benchmarking of file servers is protocol specific. Therefore, generic parameters need to be mapped to file system or protocol specific parameters depending on the system or component being benchmarked. The benchmark provides the utilities required to perform this mapping. This should allow the comparison of the results from a benchmark run on Windows with that of one on a UNIX system.

The workload mapping is performed during the load generator initialization phase. Although it will consume more CPU cycles by having each load generator perform this mapping when the benchmark master could perform it before invoking the load generators it effectively doesn't save any time by having the benchmark master do the workload mapping. The approach taken keeps any detailed knowledge about processing workload descriptions out of the benchmark master.

### 3.5.3 File Request Clustering

File request clustering is one of those things that are easier said than done. The issues raised when doing file clustering are:

- Determining operation clusters from workload traces and what the appropriate delays are between the operations.

- Determining how to generate a cluster of operations within the constraints of the file operation mix.
- Validating the cluster operation mix as it is possible to specify cluster operation mixes that will not allow the overall target mix for the workload to be obtained. Some error checking is required.

The approach taken in this benchmark is to first pick operation clusters to run, making sure that one operation cluster runs on the same thread, and then picking other operations to fill in making sure that the target operation mix is achieved.

### 3.5.4 File Sharing

A workload description can specify whether file sharing will take place. There are two types of file sharing that will be covered. The first is directory file sharing where two or more load generators will perform operations on the files in the same directory on the file system. The files themselves will not be shared between the load generators but the directory will be shared. Figure 2 below shows an example of directory sharing. Directory A and B are shared between all the load generators but the files within the directories are owned by a particular load generator. The second type of file sharing is file sharing where all the load generators will share the contents of the file. This means that either the entire file is locked (share lock) or a region of the file is locked (record lock). The load generators may try to access any region of the file. Files can be shared read-only where many load generators can access the file but none of them can write the file, in effect simulating a read-only file system. No file locking is required for files that are shared read-only. Files can be shared read-write where many load generators can access different regions, the same regions, or overlapping regions of the file. File locking must be used for this case and the underlying file system must support file locking.

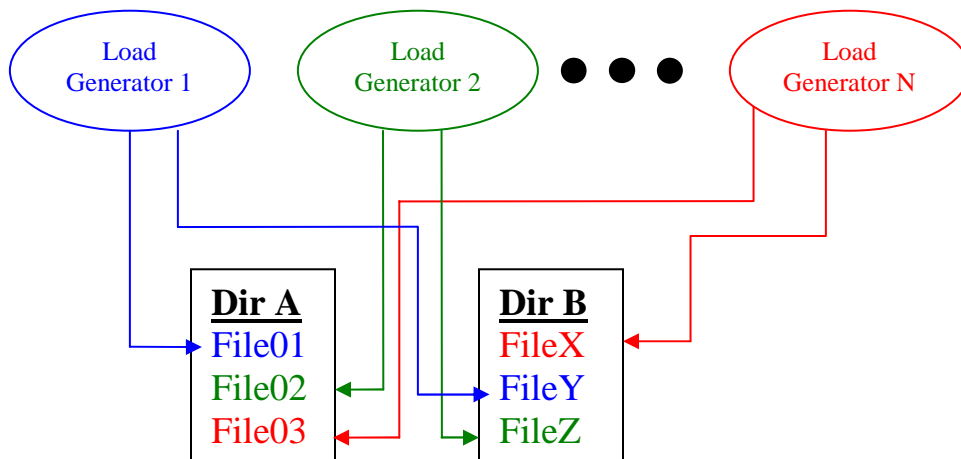


Figure 2: Directory File Sharing

### **3.6 Benchmarking Protocol**

The benchmark protocol specifies the messages sent between the benchmark master and the load generators. The protocol is message based and has a well defined OS independent functional interface which allows the OS specific details to be hidden to help portability. The OS dependent part of the implementation is based on sockets since that is available on all the target operating systems.

The protocol definition is simple assuming that a significant majority of all messages will get to their destination and that errors will be very infrequent. The protocol does error processing and attempts recovery when possible and if recovery is not possible then the benchmark run will be stopped. All messages include check data.

The protocol messages follow:

#### Phase Change

This is a master initiated message. The master specifies to the load generator the start and end time for the phase. Any phase specific information is included in the message. The load generator sends back an ACK message.

#### Get Status

This is a master initiated message. The master requests current status from the load generator. The load generator sends back status.

#### Collect Results

This is a master initiated message. The master requests the load generator to send results to the master. The load generator sends back the results.

#### Error Notification

This message could be initiated by anyone. The sender specifies the type of error condition that occurred. The receiver of this message sends back an ACK message.

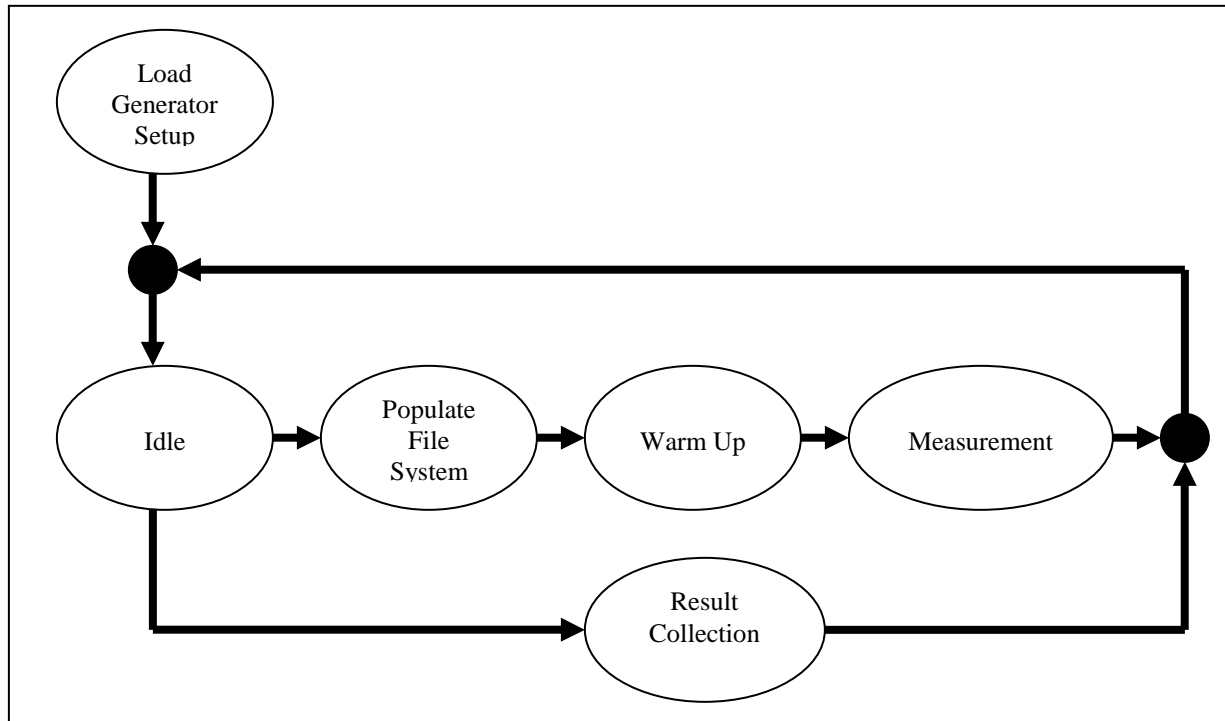
#### ACK

The sender is acknowledging receiving a message.

### **3.7 Benchmark Phases**

This section will cover the phases of the benchmark which are diagrammed in Figure 3 below. It is critical that all load generators are stepped through the benchmarking phases in the proper sequence.

NOTE: A benchmark run consists of running the following phases: *populate file system*, *warm up*, and *measurement phases*.



**Figure 3: Load Generator Phases**

### 3.7.1 Load Generator Setup

The load generator phase is initiated when the load generator program is executed. The following actions are taken during this phase:

- Verify that the benchmark software has all the components required to run.
- Validate the HW and SW setup
- Map the generic file system workload into the specific file system workload requested by the benchmark description.

The next phase is *idle*.

### 3.7.2 Idle

The component in this state is ready for a new benchmarking run or to return results and is awaiting the next phase change command from the master. The next phase is either *populate file system* or *result collection*.

### 3.7.3 Populate File System

This phase of the benchmark creates the files on the file system that are required for the load generator to run the specified workload against the file system. How the file system is populated can vary from workload to workload. The next phase is *warm up*.

### 3.7.4 Warm Up

The warm up phase is used to warm up system caches on the SUT. This phase can also be used for file system aging as it will perform operations that will cause file allocation and de-allocation. This phase attempts to run the same workload at the same rate as specified for the measurement phase. The word attempt is used because the requested load may be beyond the saturation point of the component being benchmarked resulting in an actual load less than the requested load. The next phase is *measurement*.

### 3.7.5 Measurement

The measurement phase attempts to run the workload at the rate specified in the benchmark description and measure the performance. The word attempt is used because the requested load may be beyond the saturation point of the component being benchmarked resulting in an actual measured load less than the requested load. The load generators in this phase need to use as little resources as possible to minimize the impact the benchmark has on response time and also to minimize the impact the benchmark will have measured local file system performance. The next phase is *idle*.

### 3.7.6 Result Collection

The result collection phase is where the benchmark master collects results from all load generators and then prepares. The next phase is *idle*.

## 3.8 **Benchmark Installation**

It would be nice if the benchmark package would automatically distribute itself to all the computer systems participating in the benchmarking process but nowadays with increased computer security the default settings on most computer systems make automatic distribution difficult. Currently the user must install the benchmark package on all participating computer systems. The benchmark package will register the load generator as a networking service. That will allow the master program to start up the load generator just by sending it a message.

## 4.0 **Workload**

One common complaint from end-users about the SPECsfs® benchmark has been the use of only one workload. There is a desire for a family of workloads similar to what has been done with the TPC benchmark. It is hoped that the mechanisms in the FSSB benchmark not only allow for multiple workloads and workload families but encourage the development of a wide variety of workloads.

The specification of workloads used by FSSB will be covered in another paper.

## 5.0 Terminology

A few key terms are defined here to ensure the reader understands how these terms are being used in this paper.

### Component Level Benchmark

A component level benchmark would measure the performance of one component of the system level solution. For example that could be the performance of an NFS Client, CIFS Client, NFS Server, or a CIFS Server. Given this definition, then an NFS Server benchmark like SPECsfs® would be considered a component benchmark that only benchmarks NFS servers. It does not benchmark the NFS client.

### Load Generator

A load generator generates load against the component or system being benchmarked. A load generator is a single process running on a computer system that may spawn many threads and generates a particular type of load.

### System Level Benchmark

A system level benchmark would measure the performance of an entire system. It can run on just one computer to measure the performance of the local file system or it can run on hundreds of clients using many servers.

### SUT

The term **System Under Test** (SUT) is used to refer to the entire system being tested [Jain91].

## 6.0 References

- [Aran04] A. Aranya, C. Wright, and E. Zadok, "Tracefs: A File System to Trace Them All", *Proceedings of 2004 USENIX Conference on File and Storage Technologies*, April 2004.
- [Grib98] S. Gribble, G. Manku, D. Roselli, E. Brewer, T. Gibson, and E. Miller, "Self-Similarity in File Systems", *Proceedings of the 1998 Sigmetrics Conference*, pp. 141-150, June 1998.
- [Jain91] R. Jain, "The Art of Computer Systems Performance Analysis", John Wiley & Sons, Inc., 1991.
- [Rose00] D. Roselli, J. R. Lorch, and T. E. Anderson, "A Comparison of File System Workloads", *Proceedings of 2000 USENIX Annual Technical Conference*, June 2000.
- [Smit97] K. A. Smith, and M. I. Seltzer, "File System Aging - Increasing the Relevance of File System Benchmarks", *Proceedings of SIGMETRICS 1997: Measurement and Modeling of Computer Systems*, pp. 203-213, June 1997.
- [Smit01] K. A. Smith, "Workload-Specific File System Benchmarks", *Thesis*, Harvard College Cambridge Mass, January 2001.
- [SPEC01] "SPEC SFS 3.0 Whitepaper 1.0", Standard Performance Evaluation Corporation, Warrenton, Virginia, 2001.
- [Tang95] D. Tang, "Benchmarking Filesystems", *Thesis, TR-19-95* Harvard College Cambridge Mass, April 1995.
- [Voge99] W. Vogels, "File System Usage in Windows NT 4.0", *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pp. 93—109, 1999.