Storage Performance Engineering

The Unofficial DSTAT Reference Manual

DSTAT Revision 00.06
Manual Revision May 1, 1998

Ken Bates
Compaq Computer Corporation
Ken.Bates@Compaq.com

Contents

Introduction

DSTAT, which was introduced with version 3.0 of the HSx[1] Operating System Firmware (HSOF), is an HSx utility designed to report detailed performance information about the controller. This information, which is updated on a periodic basis, is intended to be passed to a host based program (that will be developed later) that will interpret and analyze the DSTAT data, highlighting any potential bottlenecks within the I/O subsystem, and possibly making recommendations to improve performance.

Although intended primarily for computer based analysis, the output from DSTAT is in plain ASCII text, with well defined fields and delimiters. As such, the data may also be captured by a host based command procedure and analyzed via a user written program or spreadsheet. Indeed, it is even possible for a human operator to view the DSTAT data and interpret it, constructing detailed performance information for the units and controllers, as well as being able to draw conclusions as to which configuration options will improve performance.

Since the individual DSTAT fields may appear somewhat cryptic, and since much of the important performance information is based on calculations using these fields, human interpretation of DSTAT data may be somewhat difficult. This manual has been written to address that problem by explaining the format of the DSTAT data, the meanings of each of the individual fields, the suggested ancillary calculations to yield additional performance information, and suggested courses of action to improve performance under a variety of conditions.

It should be noted that this manual is, as the title suggests, an “unofficial” guide. Lacking any official DSTAT documentation, it is arguably better than nothing, but it is nonetheless a part time effort. It has been written to address the needs of customers and field support and to assist in evaluating and tuning the HSx. Any errors or omissions in this document should be attributed to the author, and not to the Performance Engineering group, the HSx support team, or Digital Equipment.

If you see this symbol in the manual, it indicates a bug (or unexpected feature) in DSTAT.

DSTAT Theory of Operation

DSTAT consists of two independent components: internal code in the HSOF that is always present, and the HSx based DSTAT utility, which is run on demand via the CLI.

HSOF Modifications

In order to collect performance information, the HSOF was modified to collect “interesting” performance information for each unit[2] on the HSx. The information collected was required to meet two criteria (in addition to being possible to collect in the first place):

[1]Within this manual, the term “HSx” collectively refers to the HSD, HSJ, and HSZ series controllers.

[2]A “unit” may be a disk drive, an array, or a partition created with CLI commands.

1. The minimum amount of information necessary for performance analysis, and
2. Little or no additional computational overhead on the HSx processor.

Both criteria were chosen so that the performance impact of including DSTAT functionality would be minimal. As an example, instead of maintaining a counter for the cache hit rate (requiring two increments and one floating point division for each unit), integer counters are instead maintained for total requests and total cache hits received to date (requiring only two increments). It is then the responsibility of the analysis program to divide the total cache hits by the total requests, resulting in the cache hit rate. Although the saving of a single floating divide instruction may seem to be a trivial exercise, it should be remembered that the HSx does not have a floating point processor, so any floating point calculations become quite expensive. This is important since derived performance information may require many dozens of divides per unit. Additionally, there may be as many as 128 (virtual) units per HSx controller. Finally, some performance calculations involve the use of exponentials, which would clearly be a drain on the resources of the HSx processor. Given that a certain minimal set of calculations would have to be performed by the analysis program, it is a simple extension to require that all calculations be done by that program, and that the HSOF provide only the necessary data to effect those calculations.

DSTAT Operation

The counters maintained by the HSOF are continuously being updated as I/O activity takes place. Since these counters are internal to the HSOF, the DSTAT utility was developed to access and output them to the user.

In operation, DSTAT prompts for an iteration interval, then outputs the following ASCII data to the (possibly virtual) terminal:

- A single line providing global controller information,
- A single character identifying the type of page (additional information may follow this single character, depending on the specific page),
- For each page, one or more lines providing detailed information,
- An end of page indicator, and
- An end of data indicator.

DSTAT will then wait for the iteration interval to elapse, then repeat the above sequence.

A “page” as defined by DSTAT is a grouping of particular information. This page may consist of repetitive lines of information (one line per unit, as an example), or information that varies from one line to the next (global controller information).

The intent is to have an analysis program on a host computer that connects to the HSx and launches DSTAT. This program will then accept and interpret the data that DSTAT sends. With a suitably intelligent algorithm, the analysis program will be able to analyze the performance of the HSx, pinpoint potential bottlenecks, and make suggestions for configuration changes that may improve performance. Indeed, a more advanced program would be able to send CLI commands to the HSx that would dynamically reconfigure the controller in response to changing workload conditions. If this program were to have embedded heuristics, it would be able to gradually tune the HSx controller, learning from its mistakes, and dynamically adapting to ensure that all configuration parameters were optimal for existing conditions.

Since such a program is not currently available (nor is likely to be in the near future, given current resource limitations), this manual is intended to provide sufficient information to allow an operator to interpret and analyze all current DSTAT data. Where appropriate, suggestions will be made for performance tuning.

DSTAT Output

Since DSTAT is nothing more than a utility program on the HSx, all that is needed to invoke DSTAT is a simple CLI "RUN" command. The output data from DSTAT exceeds 80 columns, so 132 column mode should be selected on the terminal. The exact procedure used to connect will vary, depending on whether a direct or virtual terminal connection is made, the HSx type (HSD, HSJ, or HSZ), and the operating system in use. All examples in this manual were obtained from an HSJ-50 (CI based).

Invoking DSTAT

Once connected to the HSx and at the CLI prompt, simply run the DSTAT program:

```
HSJ8>RUN DSTAT
```

When DSTAT begins running, it will first identify itself:

```
DSTAT [HSJ50 Version 00.06] (19-Feb-96)
```

The data within brackets identifies the type of controller (HSJ50 in the above example), as well as the software version of DSTAT (00.06 in the example). The software build date is shown within parentheses (19-Feb-96 in the example). The version number is divided into two subfields separated by a decimal point ("00" and "06" in the example). The first field (00) represents a major functionality release, while the second field (06) represents a bug fix or minor enhancement.

Automated analysis programs should always check the first field against what they expect. If this field is different than expected, it indicates that a major functionality change has occurred in the program, such as additional information in the output data, changed output data, etc.

Answering DSTAT Prompts

Once the preliminary DSTAT information has been printed, DSTAT will prompt for a scan interval:

```
Interval (seconds) [60]?
```

At this prompt, enter the number of seconds between DSTAT data scans. This number represents the amount of time DSTAT will pause before examining and outputting the contents of the counters maintained by the HSOF.

The value of the interval is important, since any subsequent analyses will be affected by it. If the value is too large, then variations in the workload will be masked. On the other hand, if the value is too small, massive amounts of data may be collected, with each sample being similar to the next. Only experience will determine the best value. In general, the idea is to have the interval set such that instantaneous variations will be masked by the length of the interval, but long term variations will be picked up. A good starting point is the default value of 60 seconds.

If an interval of 0 is entered, DSTAT behaves somewhat differently. Instead of using the interval as the time between scans, DSTAT will output the data pages (described later) only once, zero all counters, then exit. This behavior is intended for those programs that wish to run DSTAT on a periodic basis, collecting information "when it is convenient", rather than at fixed intervals.

There is a bug in DSTAT version 00.06 that does not zero the counters when a zero interval is specified.

Next, DSTAT will prompt for the data pages:

```
Pages? P
```

This prompt is present to allow for future DSTAT enhancements. DSTAT has been designed as a general purpose data collector that will output a variety of internal controller and unit information. Each logical grouping of information is referred to as a "page". At present, the only supported page is the performance information page. Therefore, the only legal response to the page prompt from DSTAT is "P". Note that this prompt *is* case sensitive. A response of "p" is *not* the same as a response of "P".

As additional pages of information are added to DSTAT, this manual will be updated to reflect the optional responses.

DSTAT Data

After the program prompts have been answered, DSTAT will output data from the HSx counters. This output follows a fixed format consisting of an interval header, followed by a field header and field data lines for each page. A pseudo-code example of the DSTAT output process is as follows:

```
repeat
   output intervalHeader
      repeat for all pages
         output pageIdentifier
         output page specific information
      end repeat
   wait scanInterval
end repeat
```

DSTAT Interval Header

At the start of an information page printout (when the time interval has expired), DSTAT will output a single line of information that provides global controller information:

```
HSJ50 V50J-0  29-AUG-1996 12:14:08.9  87.5% Idle
```

There is a bug in DSTAT version 00.06 that does not print the interval header line when DSTAT outputs the first data page. As a result, the date and exact time of the first DSTAT collection is unknown, making determination of the exact time interval from the first to second DSTAT page only an approximation (based on the specified interval).

The interval header consists of five discrete fields, with each field separated by a tab character.

There is a bug in DSTAT version 00.06 that the interval header fields are not separated by tabs. Instead, they are separated by one or more space characters.

The interval header fields are:

Controller Type

This field contains the type of controller DSTAT is running on ("`HSJ50`" in the example). It will remain the same from one interval to the next, and is present to allow each page output to contain sufficient information to identify the controller.

HSOF Firmware Version

This field contains the current firmware version ("`V50J-0`" in the example). Like the controller type, this field will be unchanged from one interval to the next, and is present for the same reason.

Scan Date

The scan date field ("`29-AUG-1996`" in the example). This is the date when DSTAT scanned the counters collected by the HSOF.

Scan Time

The scan time field ("`12:14:08.9`" in the example). This is the time of day when DSTAT scanned the counters collected by the HSOF. The scan interval time may be determined by subtracting the time from the previous DSTAT page from this time. The resulting value should be quite close to the specified scan interval, but may vary somewhat due to higher priority operations (such as I/O). When calculating the elapsed time, care should be taken to check the scan date field, since the dates will switch at midnight.

HSx Idle Time

The idle time field ("`87.5% Idle`" in the example) contains the percentage of time that the HSx was idle during the scan interval. Idle time is defined as the time that the HSx was not performing I/O or other operations (such as utilities). The utilization of the HSx may be obtained by subtracting this value from 100%. In the example given, the utilization of the HSJ50 was 12.5%.

DSTAT Performance Page

At present, the only supported page in DSTAT is the unit performance page. This page consists of the "P" character as the first character (to identify the performance page) followed by field identification information on the same line. Following this line will be one line for each user visible unit known to the controller. Each field of the unit data lines contains information specified in the field header.

DSTAT Field Header Line

Following the interval header, DSTAT will print out the field header line:

```
P  Unit Stat  RdCmd  Cnt / RdQ  RdBlks  RdHits  CachBlks  RdPrg  WrCmd  Cnt / WrQ  WrBlks  WrPrg
```

This line consists of text that identifies the contents of each subsequent field for all lines on the performance page. This line is present primarily for human identification, and may also be used as the title field for inclusion into a spreadsheet or database. Each field label has been condensed for space reasons.

The contents of most of the individual fields (described later) are accumulating. That is, when DSTAT outputs a data line, it will output the contents of the counters at the instant of the scan. Once DSTAT has completed its scan, further I/O operations will continue to increase the values of the counters. Thus, a host based program must subtract the values from the previous DSTAT scan from the values obtained in the current scan to determine the amount of change in the counters over the scan interval.

The meanings of the individual fields are as follows:

P

This field indicates that the field header line (and subsequent data lines until the [EOP] indicator) are for the performance page. If additional pages are added to DSTAT in future versions, this character will reflect which page is being printed.

Unit

The user visible unit number of the device. Note that this unit number may represent a single disk, multiple disks (in the case of an array), a portion of a disk (in the case of a partitioned disk), or a portion of multiple disks (in the case of a partitioned array).

Stat

A two character ASCII field indicating the status of the device. Possible values are:

R Read caching is enabled for the unit..

r Read caching is *not* enabled for the unit.

W Writeback caching is enabled for the unit.

w Writeback caching is *not* enabled for the device.

As an example, if this field consisted of “Rw”, then it is an indication that read caching is enabled for the unit, but writeback caching is *not* enabled. Note that this may change from one scan to the next if the unit’s cache status changes.

RdCmds

The total number of read commands received to date. The value of this field will increase from one scan to the next if read commands are received.

Cnt / RdQ

These labels identify two separate but related fields, both of which must be taken together to produce meaningful information. Within the HSx, code exists that runs background tasks periodically (several times per second). One of these tasks samples the number of outstanding read commands that exist for each unit at that point in time. This number is accumulated into the **RdQ** location. The number of times that the read queue on the unit was sampled is loaded into the **Cnt** location. When DSTAT prints these values, the **RdQ** location represents the total of all outstanding read commands seen on a unit, while the **Cnt** location represents the number of times the read queue was sampled. Although the **Cnt / RdQ** is one label, the two fields (*Cnt* and *RdQ*) are two distinct fields, separated by a tab character.

As an example, assume that **Cnt** had a value of 72, and **RdQ** had a value of 56. This means that the read queue for this unit was sampled 72 times. Each time that the queue was sampled, the number of outstanding requests was accumulated, and after 72 samples, the total number of outstanding requests seen was 56.

Note that the label **Cnt** exists in two places; once preceding the **RdQ** label, and once again preceding the **WrQ** label (covered later).

Note also that unlike most other fields in DSTAT, these fields do *not* accumulate from one scan to the next. The values of these fields represent the count and queue over the scan interval.

RdBlks

The number of 512 byte blocks of data read to date. This includes both cached and non-cached data. The value of this field will increase from one scan to the next if read commands are received.

RdHits

The number of read commands that were satisfied from controller cache. The value of this field will increase from one scan to the next if read commands are received and satisfied from cache.

CachBlks

The number of 512 byte blocks of data that were read from controller cache. The value of this field will increase from one scan to the next if read commands are received and satisfied from cache.

RdPrg

The number of 512 byte blocks of data that have been purged from cache as a result of read commands issued to this unit. Note that if data has been purged from cache as a result of I/O to a different unit, those blocks will not be reflected in this field. Only data purged from cache as a result of read commands issued to this specific unit will be tallied by this field. Similarly, data purged from cache as a result of write commands to *any* unit (including this one) are not counted in this field. The value of this field will increase from one scan to the next if data is purged from cache as a result of a read command to this unit.

WrCmd

The number of write commands received. The value of this field will increase from one scan to the next if write commands are received.

There is a bug that causes the write command count to decrement from time to time. That is, it will increase from scan to scan as expected, but occasionally, it will decrement. This bug is still being researched to find the exact circumstances causing its appearance.

Cnt / Wrq

These fields are identical to the *Cnt/RdQ* fields, except that they pertain to writes instead of reads. Note that these fields do *not* accumulate from one scan to the next. The values of these fields represent the count and queue over the scan interval.

WrBlks

The number of 512 byte blocks of data written. The value of this field will increase from one scan to the next if write commands are received.

WrPrg

The number of 512 byte blocks of data that have been purged from cache as a result of write commands issued to this unit. Note that if data has been purged from cache as a result of I/O to a different unit, those blocks will not be reflected in this field. Only data purged from cache as a result of write commands issued to this specific unit will be tallied by this field. Similarly, data purged from cache as a result of read commands to *any* unit (including this one) are not counted in this field. as a result of write commands to *any* unit (including this one) are not counted in this field. The value of this field will increase from one scan to the next if data is purged from cache as a result of a write command to this unit.

Unit Information Lines

Immediately following the field header lines will be zero or more unit information lines. Each information line represents the collected data from one specific unit (reflected in the unit field). If no units are known to the controller, there will be no unit information lines.

There is a bug in DSTAT version 00.06 that occurs when any field exceeds the predefined field width. If this occurs, asterisks (***) will be printed for that field value. This is a result of the internal HSOF accumulating the values without being able to reset them, which will eventually result in numbers that overflow their printing field width. At present, the only workaround is to exit DSTAT and run VTDPY, then exit once VTDPY begins execution. When VTDPY runs, all internal HSOF counters will be set to zero. Following this step, run DSTAT again, remembering that all counters had been cleared by VTDPY, so they will be much lower than the last DSTAT scan. A good practice is to always run VTDPY first (thereby clearing the counters), then run DSTAT.

End Of Page

Immediately following the last unit information line will be an end of page indicator:

[EOP]

This is an indication that all information for the current page has been printed. If there is more than one page type (not currently implemented), the next page will follow the *EOP* indicator.

End Of Data

When all pages for a given scan interval have been printed, DSTAT will output an end of data indicator:

[EOD]

Since the current implementation of DSTAT only supports the performance page, this indicator will immediately follow the end of page indicator:

[EOP]
[EOD]

The EOD indicator is a signal to the analysis program that all data for a give scan interval has been output, and that no more data will be sent until the scan interval has once again elapsed.

There is a bug in DSTAT version 00.06 that when the [EOD] indicator is ouput, no return character (hex 0D) follows it. The result of this is that the "cursor" is left immediately following the [EOD], making it difficult for analysis programs that read lines of ASCII text from DSTAT to determine when output has completed.

Example DSTAT Output

The following is an (abbreviated) example of a DSTAT run. This example shows two DSTAT scans. In this example, only four units are shown. All other units have been eliminated from the printout for clarity.

```
HSJ50 V50J-0  10-SEP-1996 12:32:54.2  11.8% Idle
P  Unit Stat  RdCmd    Cnt / RdQ  RdBlks     RdHits     CachBlks  RdPrg  WrCmd   Cnt / WrQ  WrBlks    WrPrg
   101  Rw   1459241  61  113    8001412    1251538    4707472   54421  141501  61    0    1731629   61194
   102  Rw   1972059  61  156    11050491   1817129    8351439   9070   117167  61    0    1484776   58345
   103  Rw   1440602  61  299    7805524    1231072    4494462   49903  134072  61    0    1689954   26639
   104  Rw   1414230  61  201    7701252    1215996    4447671   52563  110600  61    0    1528037   30101
[EOP]
[EOD]
HSJ50 V50J-0  10-SEP-1996 12:33:55.3  60.0% Idle
P  Unit Stat  RdCmd    Cnt / RdQ  RdBlks     RdHits     CachBlks  RdPrg  WrCmd   Cnt / WrQ  WrBlks    WrPrg
   101  Rw   1461738  61  144    8003909    1254035    4709969   54421  141501  61    0    1731629   61194
   102  Rw   1974556  61  151    11052988   1819626    8353936   9070   117167  61    0    1484776   58345
   103  Rw   1443099  61  435    7808021    1233569    4496959   49903  134072  61    0    1689954   26639
   104  Rw   1416727  61  271    7703749    1218493    4450168   52563  110600  61    0    1528037   30101
[EOP]
[EOD]
```

In this example, although the specified scan interval was 60 seconds, the difference between the first and second scan times (*Time*) was 61.1 seconds. The work performed in the scan interval (read commands, read hits, etc.) may be calculated by subtracting the values for each unit in the first scan from the value for the same unit in the second scan. As an example, there were a total of 1,459,241 read commands for unit 101 in the first scan, while the second scan shows that there were a total of 1,461,738 read commands. This is a difference of 2,497 read commands, or a rate of 40.87 read requests per second for unit 101 (2497 / 61.1).

DSTAT Performance Page Implicit Data

Although the amount of performance information output by DSTAT may seem rather minimal at first glance, it is possible (and intended) that the analysis program manipulate this data to obtain further information. Most calculations are quite straightforward for a program, albeit somewhat cumbersome for a human to perform in real time. The following list of data that is obtainable from simple calculations on the provided DSTAT

data is not intended to be a comprehensive list, but does represent information that should be quite useful to the performance analysis of the HSx.

Since the objective of obtaining this information is to analyze performance (covered later in this manual), all derived data will be given arbitrary (and rather cryptic) names that will be referenced later.

Note that in the equations that follow, a program must take care to verify that any denominators are non-zero, or a divide by zero error will occur. Note also that all calculations that involve read cache will only be valid if the unit status is "R". Similarly, any writeback cache calculations are only applicable if the unit status is "W".

Some calculations require only the data from a specific unit, others may require the summation of data from all units, while others may require both individual and summation data.

In the following calculations, if a variable name is prefixed with a "∑" symbol, then it means that the values for *all* units should be summed. As an example, *RdCmds* is the value for the number of read commands issued to a single unit, while *∑RdCmds* is the sum of all read commands issued to all units. Thus, the expression "*RdCmds / ∑RdCmds*"would represent the fraction of total controller read commands seen by a specific unit.

Finally, with the exception of *Cnt / RdQ* and *Cnt / WrQ*, the calculations use delta, not absolute values. As an example, referring to the previous DSTAT printout, any calculations involving *RdCmds* should use 2,497, since that represents the delta in *RdCmds* between successive DSTAT scans.

Global Data

Some data, such as the total number of commands through the entire controller, are not specific to any one unit. They may, however, require summations of all units on the controller.

Time

This is the value in seconds of the scan interval. It is obtained by taking the scan time field (from the DSTAT interval header), and subtracting from it the scan time field from the *previous* DSTAT interval header. Note that this implies that for the first scan, it is not possible to calculate this value. The calculated value should be close, but not necessarily in agreement with, the interval time specified in the DSTAT prompt.

NumUnits

This value represents the total number of units as implicitly reported by DSTAT via the number of unit information lines. It may vary from a low of 0 to a high of 128 units with version 3.0 of the HSOF.

CtlrData

(∑RdBlks + ∑WrBlks) / Time / 2

This is the data rate in KB/Sec for all commands processed by the controller. Note that a KB is considered to be 1,024 bytes.

CtlrHitRate

∑RdHits / ∑RdCmds / Time

This represents the overall controller read cache hit rate. It will vary from zero (no hits by any unit) to one (all read requests for all units satisfied from read cache).

CtlrMissData

(∑RdBlks - ∑CachBlks) / Time / 2

This represents the average read data rate in KB/Sec through the controller for all requests that were satisfied from the units (and not from controller read cache).

CtlrMissRate

(∑RdCmds - ∑RdHits) / Time or
1 - CtlrHitRate

This represents the average controller read request rate throught the controller that were sent to the units (and not to controller read cache).

CtlrQue

CtlrRdQue + CtlrWrQue

This is the average queue length for the controller.

CtlrRate

(∑RdCmds + ∑WrCmds) / Time

This is the request rate in requests per second for all commands processed by the controller.

CtlrRdCachData

(∑CachBlks) / Time / 2

This represents the average read cache data rate through the controller in KB/Sec. Note that a KB is taken to mean 1,024 bytes.

CtlrRdCachRate

(∑RdHits) / Time

This represents the average read cache request rate through the controller.

CtlrRdCmdPcnt

(∑RdCmds) / (∑RdCmds + ∑WrCmds)

This represents the fraction of all commands received by the controller that are read commands (sometimes mistakenly referred to as the read/write ratio).

CtlrRdData

(∑RdBlks) / Time / 2

This represents the average read data rate (in KB/Sec) through the controller. Note that a KB contains 1,024 bytes.

CtlrRdDataPcnt

(∑RdBlks) / (∑RdBlks + ∑WrBlks)

The fraction of all data transferred by the controller that are as a result of read commands.

CtlrRdHitSize

∑CachBlks / ∑RdHits

The average size in 512 byte blocks of a read cache hit on the controller.

CtlrRdMissSize

(∑RdBlks - ∑CachBlks) / (∑RdCmds - ∑RdHits)

The average size in 512 byte blocks of a read miss (data that is read from disk) throught the controller.

CtlrRdQue

∑(RdQ / Cnt)

This value represents the average read queue length for the controller. Note that the **Cnt** variable is the one associated with the **RdQ** variable, *not* the one with the same name associated with the **WrQ** variable. Note also that unlike test programs such as IOX and UIOX, queue lengths are expressed with the standard mathematical meaning, e.g. a queue length of one means one request is in the queue (where IOX and UIOX translate a value of one to mean the *two* requests are outstanding).

Note also that this value is based on statistical sampling, so errors may be introduced when the value of **Cnt** is small (implying a small scan interval specified to DSTAT).

CtlrRdRate

(∑RdCmds) / Time

This represents the average read request rate through the entire controller.

CtlrRdResp

CtlrRdQue / (∑RdCmds / Time)

This represents the average response time in seconds for all read commands processed by the controller.

CtlrRdSize

∑RdBlks / ∑RdCmds

The average size in 512 byte blocks of read data transferred by the controller.

CtlrResp

*CtlrRdResp * CtlrRdCmdPcnt + CtlrWrResp * CtlrWrCmdPcnt*

This is the average response time in seconds of all I/O commands sent to the controller.

CtlrWrData

(∑WrBlks) / Time / 2

This represents the average write data rate (in KB/Sec) through the controller.

CtlrWrQue

∑(WrQ / Cnt)

This value represents the average write queue length for the controller. Note that the **Cnt** variable is the one associated with the **WrQ** variable, *not* the one with the same name

associated with the **RdQ** variable. Note also that unlike test programs such as IOX and UIOX, queue lengths are expressed with the standard mathematical meaning, e.g. a queue length of one means one request is in the queue (where IOX and UIOX translate a value of one to mean the *two* requests are outstanding).

Note also that this value is based on statistical sampling, so errors may be introduced when the value of **Cnt** is small (implying a small scan interval specified to DSTAT).

CtlrWrRate

(∑WrCmds) / Time

This represents the average write request rate through the entire controller.

CtlrWrResp

CtlrWrQue /(∑WrCmds / Time)

This represents the average response time in seconds for all write commands processed by the controller.

CtlrWrSize

∑WrBlks / ∑WrCmds

The average size in 512 byte blocks of write data transferred by the controller.

Unit Data

Some data is specific to a single unit, and may be calculated without reference to any other unit's data. An example might be the total number of read requests, *RdCmds*.

UnitData

(RdBlks + WrBlks) / Time / 2

The total data rate for the unit (both read and write commands) in KB/Sec.

UnitQue

UnitRdQue + UnitWrQue

The average queue length for the unit.

UnitRate

(RdCmds + WrCmds) / Time

The total request rate for the unit (both read and write commands) in requests per second.

UnitRdCachData

CachBlks / Time / 2

This is the read data rate in KB for all data that was supplied from read cache.

UnitRdCachRate

RdHits / Time

This represents the average number of read requests per second satisfied by cache for this unit.

UnitRdCmdPcnt

RdCmds / (RdCmds + WrCmds)

This read command fraction for the unit (sometimes incorrectly referred to as the read/write ratio). It will vary from zero (all writes) to one (all reads).

UnitRdData

RdBlks / Time / 2

This is the read data rate in KB for all data for the unit. Note that in this example (and those that follow), a KB is defined as 1,024 bytes.

UnitRdDataPcnt

RdBlks / (RdBlks + WrBlks)

This is similar to **RdCmPcnt**, but instead calculates the fraction of data for this unit that were reads. It may vary from zero to one.

UnitRdHitSize

CachBlks / RdHits

This value is the average size in 512 byte blocks of all commands satisfied from read cache over the last sample interval.

UnitRdMissSize

(RdBlks - CachBlks) / (RdCmds - RdHits)

This figure is the average size of read requests that were *not* satisfied from cache. In other words, it is the average size of all read cache miss commands.

UnitRdSize

RdBlks / RdCmds

This value is the average read size in 512 byte blocks over the last sample interval. It is the average of *all* read commands, both cache hits and cache misses.

UnitRdHitRate

RdHits / RdCmds

This figure is the read cache hit rate, expressed as a fraction from zero (no hits) to one (100% read cache hits).

UnitRdMissData

(RdBlks - CachBlks) / Time / 2

This is the read data rate in KB for all data supplied from disk. In other words, the read miss data rate.

UnitRdMissRate

(RdCmds - RdHits) / Time

This represents the number of read requests per second that were not satisfied from cache, e.g., the number of read requests that were sent to disk.

UnitRdQue

RdQ / Cnt

This value represents the average read queue length for the unit. Note that the **Cnt** variable is the one associated with the **RdQ** variable, *not* the one with the same name associated with the **WrQ** variable. Note also that unlike test programs such as IOX and UIOX, queue lengths are expressed with the standard mathematical meaning, e.g. a queue length of one means one request is in the queue (where IOX and UIOX translate a value of one to mean the *two* requests are outstanding).

Note also that this value is based on statistical sampling, so errors may be introduced when the value of **Cnt** is small (implying a small scan interval specified to DSTAT).

UnitRdRate

RdCmds / Time

This represents the average number of read requests per second sent to this unit. It includes both cache hits and misses.

UnitRdResp

UnitRdQue / (RdCmds / Time)

This value represents the average read response time (measured in seconds) from the unit. Since it is a derived value based on a sampling interval, there is a possibility of errors if the number of samples is small (see **RdQue**) or the value of **RdCmds** is small.

UnitResp

*UnitRdResp * UnitRdCmdPcnt + UnitWrResp * UnitWrCmdPcnt*

The average response time for all I/O commands sent to the unit.

UnitWrData

WrBlks / Time / 2

This represents the average write data rate in KB/Sec for the unit.

UnitWrQue

WrQ / Cnt

This value represents the average write queue length for the unit. Note that the **Cnt** variable is the one associated with the **WrQ** variable, *not* the one with the same name associated with the **RdQ** variable. Note also that unlike test programs such as IOX and UIOX, queue lengths are expressed with the standard mathematical meaning, e.g. a queue length of one means one request is in the queue (where IOX and UIOX translate a value of one to mean the *two* requests are outstanding).

Note also that this value is based on statistical sampling, so errors may be introduced when the value of **Cnt** is small (implying a small scan interval specified to DSTAT).

UnitWrRate

WrRate / Time

This represents the average write request rate for the unit.

UnitWrResp

UnitWrQue / (WrCmds / Time)

This value represents the average write response time (measured in seconds) from the unit. Since it is a derived value based on a sampling interval, there is a possibility of errors if the number of samples is small (see **WrQue**) or the value of **WrCmds** is small.

UnitWrSize

WrBlks / WrCmds

The average write size in 512 byte blocks for the unit.

Fractional Data

Some data, such as the fraction of overall controller commands contributed by a single unit, require both unit specific data and the summation of all units on the controller.

UnitCachCmdPcnt

RdHits / ∑RdHits

This is the fraction of all read cache hits on the controller done by this specific unit.

UnitCachDataPcnt

CachBlks / ∑CachBlks

This is the fraction of all read cache data transferred on the controller done by this specific

UnitCmdPcnt

(RdCmds + WrCmds) / (∑RdCmds + ∑WrCmds)

This is the fraction of all commands on the controller done by this specific unit.

UnitCmdRatio

(RdCmds + WrCmds) / ((∑RdCmds + ∑WrCmds) / NumUnits)

This value gives an indication of the load imbalance experienced by this unit. It does so by taking the number of commands sent to this unit in relationship to the total number of commands sent to the entire controller, taking into account the total number of units on the controller. If the resulting value is 1, then it indicates that this unit is receiving its share of a perfectly load balanced system. If the value is *n*, then it indicates that this unit is receiving *n* times more I/O than it would if it were perfectly balanced (note that *n* may be less than or greater than 1).

UnitDataPcnt

(RdBlks + WrBlks) / (∑RdBlks + ∑WrBlks)

This is the fraction of all data on the controller transferred by this specific unit.

UnitDataRatio

(RdBlks + WrBlks) / ((∑RdBlks + ∑WrBlks) / NumUnits)

This value gives an indication of the data load imbalance experienced by this unit. It does so identically with the *UnitCmdRatio* variable, except that it an indication of the data rate imbalance. Note that it is possible for unit to have a high *UnitCmdRatio* and a low *UnitDataRatio* at the some time (or vice versa).

UnitRdCmdPcnt

RdCmds / ∑RdCmds

This is the fraction of all read commands on the controller done by this specific unit.

UnitRdCmdRatio

RdCmds / (∑RdCmds / NumUnits)

This value gives an indication of the read load imbalance experienced by this unit. The derivation and meaning is identical to the *UnitCmdRatio*, except that only read commands are taken into account.

UnitRdDataPcnt

RdBlks / ∑RdBlks

This is the fraction of all read data on the controller transferred by this specific unit.

UnitRdDataRatio

RdBlks / (∑RdBlks / NumUnits)

This value gives an indication of the read data load imbalance experienced by this unit. The derivation and meaning is identical to the *UnitDataRatio*, except that only read data is taken into account.

UnitRdPrgRatio

(RdPrg / ∑RdPrg) / (RdBlks / ∑RdBlks)

This value, known as the read purge ratio, is designed as a measure of read cache efficiency for a particular unit. It will be covered in detail in the analysis section of this manual.

UnitWrCmdPcnt

WrCmds / ∑WrCmds

This is the fraction of all write commands on the controller done by this specific unit.

UnitWrCmdRatio

WrCmds / (∑WrCmds / NumUnits)

This value gives an indication of the write load imbalance experienced by this unit. The derivation and meaning is identical to the *RdCmdRatio,* except that only write commands are taken into account.

UnitWrDataPcnt

WrBlks / ∑WrBlks

This is the fraction of all write data on the controller transferred by this specific unit.unit.

UnitWrDataRatio

WrBlks / (∑WrBlks / NumUnits)

This value gives an indication of the write data load imbalance experienced by write unit. The derivation and meaning is identical to the *UnitDataRatio*, except that only read data is taken into account.

UnitWrPrgRatio

(WrPrg / ∑WrPrg) / (WrBlks / ∑WrBlks)

This value, known as the write purge ratio, is designed as a measure of write cache efficiency for a particular unit. It will be covered in detail in the analysis section of this manual.

Performance Analysis

Once DSTAT has been run and data gathered, the next step is to analyze the DSTAT output data (both explicit and calculated) for performance bottlenecks and configuration enhancement possibilities.

When performing bottleneck analysis, the key is to find the device with the highest utilization. The device with the highest utilization is, by definition, the bottleneck device. It may not always be possible to bypass or replace that device, and in many cases, might not result in a significant gain in performance. To understand the reason behind this, consider the graph shown in Figure 1. This graph depicts the relationship between utilization and (normalized) response time.

With the incoming request rate at a sufficiently low value such that there is never more than one request outstanding to the I/O subsystem, the response time is equal to the service time. This value, whatever it may be, is given a value of “1” on the vertical axis.

As the workload intensity increases, queues begin to form within the I/O subsystem, and response time rises. The rise in response time is proportional to the utilization of the I/O subsystem, and is quite non-linear. As may be seen in Figure 1, the increase in response time is relatively slow at the low utilization levels, but rises rapidly once the utilization exceeds 75% or so. From this graph, it should be apparent that the goal is to ensure that all devices within the I/O subsystem are well below the 80% utilization level.

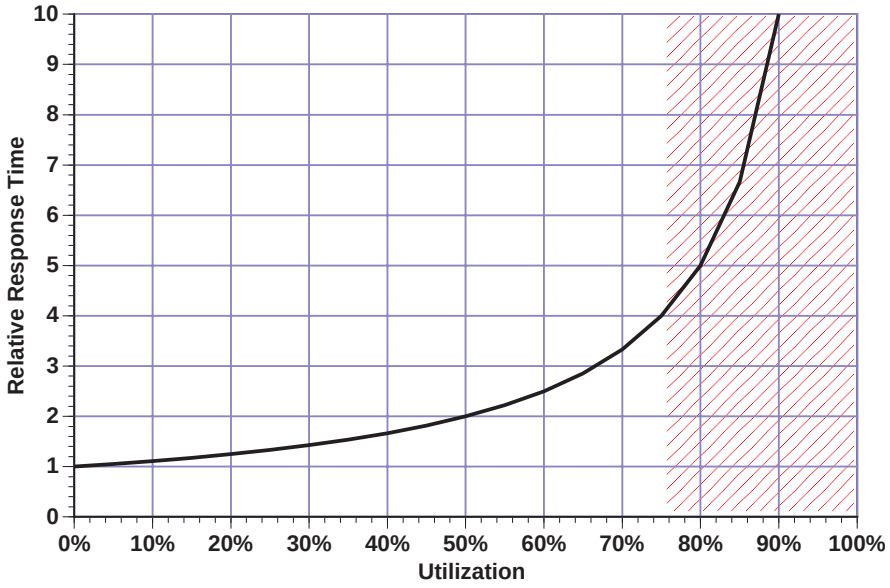

Figure 1
Response Time - Utilization Relationship

Two facts should be remembered about the utilization / response time relationship:

- Increases in response time become very significant once the utilization exceeds 75-80%. At 50% utilization, the response time is twice the no load value. At 80% utilization, the response time is five times the no load value, while at 90% utilization, the response time is ten times the no load value.
- Small variations in utilization (I/O load) have considerably more effect at high utilization levels. As an example, assume that the utilization varies ±5% due to variations in the I/O rate. If the average utilization is 20%, the response times will vary ±6% or so. At a utilization of 80% however, that same ±5% variation will cause over a 25% variation in the response time. In other words, as the utilization rises, not only does response time increase, but variations in the incoming I/O intensity cause correspondingly greater variations in the response time.

From Figure 1, it should be apparent that in no case should utilization be allowed to rise above 80%, or significant response time problems will arise. By the same token, if a device has a utilization below 30% or so, lowering the utilization of that device will have very little effect on system performance.

The second factor to take into account before beginning the analysis is that of bus bandwidth. The bandwidth specified for most busses is an electrical characteristic of the bus, and tells one the maximum rate at which data can be transferred over that bus. In practice, there is information over and above user data that is also transferred over the bus. This information, such as command and acknowledgement packets, header information, and error detection and correction information, consumes a portion of the available bus bandwidth. The user data must then fit within the remaining bus bandwidth. Because of this, the realizable maximum data rate for user data is less than the specified bus bandwidth. The exact amount varies depending on many factors, such as the bus protocol and the size of the data transfer. A general rule of thumb for most cases is that the maximum user data rate over a given bus is 85% of the bus bandwidth.

As an example, a fast narrow SCSI bus has a bandwidth of 10 MB/Sec. Using the 85% factor results in a maximum user data rate of 8.5 MB/Sec.

Remembering that to avoid excessive response times one should not exceed 80% utilization, it follows that the user data rate on a bus should not exceed 80% of maximum. Since the maximum “real data” rate is assumed to be 85% of the bus bandwidth, then a point of 80% of 85% of the bus bandwidth should be established as the desirable upper limit on user data transfer rates across a buss. Continuing with the fast narrow SCSI bus example, this works out to be 6.8 MB/Sec.

The 80% of 85% of bandwidth values are given in Table 1 for several of the popular I/O busses.

Bus	Bandwidth	Max User Rate
DSSI	4.0 MB/Sec	2.7 MB/Sec
Fast Narrow SCSI	10.0 MB/Sec	6.8 MB/Sec
Fast Wide SCSI	20.0 MB/Sec	13.6 MB/Sec
CI (Dual rail)	17.5 MB/Sec	11.9 MB/Sec
FDDI	12.5 MB/Sec	8.5 MB/Sec

Table 1
Recommended Maximum User Data Rates

It's important to remember that Table 1 gives the maximum user data rates over a bus when the objective is to minimize response time in multi-threaded I/O environment. If, on the other hand, the objective is maximum data transfer rates in a single threaded environment (such as graphics), then the 80% utilization rule may be discarded, since maximum efficiency in a single threaded environment implies 100% utilization.

Controller Analysis

Several fields should be examined when looking at the controller (in isolation from the attached disk drives).

- *HSx Idle Time* - The idle time should be greater than 20% (implying a utilization of less than 80%). If the idle time is less than 20%, then the HSx itself is contributing significantly to the system response time. The only solution to this is to offload some of the I/O operations to another controller.
- *CtlrData* - This value is the user data rate in KB/Sec. Table 1 should be consulted to determine if this value exceeds the maximum user rate shown in Table 1. If it does, the host I/O bus may potentially be a bottleneck. The only solution is to offload some of the I/O to another controller on a different host interconnect bus.
- Port utilization - In the current implementation of DSTAT, the port specific information is not explicitly available. With human assistance however, it may be possible to relate specific units (in the unit information lines) to which port they are attached to. If this is possible, then the sum of all *UnitData* fields for all units on a given port will yield the total data rate in KB/Sec for that port. Since all ports are fast narrow SCSI on the HSx controller, this value should not exceed 6,800 KB/Sec, or excessive response times may result. A potential solution for this problem is to balance the data rate across all ports by moving units. When doing so however, care should be taken to ensure that other ports do not become overloaded.

Unit Analysis

DSTAT outputs a wealth of information for each unit that may be used to improve performance. Several of the more important variables are:

- *UnitRdHitSiz* - If read caching is enabled on the unit, this value, which represents the average size of read hits, should be used to adjust the controller read cache threshold setting. Set the controller read cache threshold to the next highest integer value than this field.
- *UnitRdHitRate* - If this value drops below 20%, consider disabling cache on this unit. With a low hit rate, not only is the unit not benefiting from read caching, but the data that it is reading into cache is forcing other's data out of cache, potentially causing unnecessary read cache misses for other units.
- *UnitRdCmdPcnt* - Although not necessarily a reliable indicator of caching, very low values of read percentage indicate a low probability of high cache hit rates. Similarly, high read percentages indicate a probability of high cache hit rates (although some streams, such as sequential reads, will obtain no benefit from read cache). The primary value of this field is to "try caching" on a non-read cached

drive that shows a reasonable read percentage, monitoring the effects on unit read response time with and without read cache.

- *UnitWrSize* - If writeback caching is enabled on the unit, try setting the write cache threshold to the next higher integer value than this field.
- *RdPrgRatio* - This figure relates the amount of data purged from cache as a result of read commands to that of the entire system. In essence, it may be considered to be a “fairness” indicator. If the unit is purging data from cache exactly in proportion to the amount of data it is reading in relationship to all other units on the system, then it is assigned a value of one. If the unit is purging far less data from cache that its I/O load would indicate, then it is assigned a value less than one. On the other hand, if it is purging far more data from cache that its “fair share”, it is assigned a value greater than one.

  The larger this number, the more this unit may be “hurting” other units by potentially purging their data from cache. The smaller the number, the more this unit is being a “good citizen” by not purging other unit’s data from cache. A value of more than 10 is usually cause for considering disabling read cache on this unit.
- *WrPrgRatio* - This figure is identical to the *RdPrgRatio* variable, except it is an indication of data purged from cache as a result of a write operation. Values less than 1 indicate write overlays or write aggregation (goodness), while values greater than 1 indicate random writes that are purging other unit’s data from cache.

  The decision to disable writeback caching on a unit with a large *WrPrgRatio* value is not as easy as with reads, since the benefit of writeback cache is so great. If this value exceeds 20, then writeback caching might be discontinued on a trial basis to see the effects on response time. If this is done, *CtlrResp* should be monitored to see if the performance of the controller as a whole improves. Additionally, *UnitWrResp* should be monitored to see if the write performance of this disk decreases.
- *UnitCmdRatio* - This value, which represents the load imbalance for each unit, should be examined with an eye towards the possible creation of arrays. If this value exceeds 5 to 10 (indicating the unit is performing 5 to 10 times more I/O than it would on a perfectly load balanced system), then serious thought should be given to combining this unit with others that have a *UnitCmdRatio* less than one into a large array. If unit identity is important, then these multiple units should be combined into a single array, then that array should be partitioned to recreate the equivalent units.

Feedback

Once changes have been made, the obvious question to be asked is whether or not the changes had any significant effect. Although DSTAT outputs reams of implicit and explicit information to answer this, the best information is obtained by examining the response time fields (*CtlrResp* and *UnitResp*). If these fields have increased in value, then the attempt to improve performance has failed and the values prior to the change should be restored. If the values have decreased, then it is an indication that the changes made have effected an increase in performance.

If changes are made for read or write specific reasons, then the appropriate fields (*UnitRdResp*, *Unit WrResp*, etc.) should be referenced.

It is also important to remember that changes in response time will occur in response to changing workload conditions. Good knowledge of the workload is highly desirable, although some limited knowledge may be obtained examining unit and controller workload variables such as read sizes write sizes, cache hit sizes, etc.

Scan Interval

As mentioned earlier, the selection of the scan interval is important. If this interval is too small, not only will there be large amounts of DSTAT output to process, but minor variations in the workload will make it difficult to perform an analysis. On the other hand, excessively long scan intervals will tend to average important workload fluctuations, making it impossible to spot trends.

A good starting value for the scan interval is the default of 60 seconds. The preferred method is to perform several scans, then look for trends in the data. If the data from each scan is quite similar, then the scan interval should be increased. This may be done by forcing DSTAT to output data on demand.

Demand output is accomplished by sending DSTAT a carriage Ctrl/W character (hex 17). When DSTAT receives this character, it will output all data collected since the last data output. If an analysis program is running DSTAT, it may then dynamically adjust the scan interval by sending a Ctrl/W character to DSTAT whenever it determines it best.

There is a bug in the current version of DSTAT that outputs the demand data page incorrectly. When the performance page has completed printing, there is no [EOP] or [EOD] displayed to indicate that DSTAT has completed it's output.

Additionally, the last line of the performance page (the data output for the last unit) is not terminated with a return character. The result of this bug is that the "cursor" is left immediately following the [EOD], making it difficult for analysis programs that read lines of ASCII text from DSTAT to determine when output has completed.

This demand output can be quite useful when DSTAT is driven by a host based analysis program. As an example, the program could specify the initial scan interval to be a very large value, say 3600 seconds. Once DSTAT was started, the program would send the demand character after a 60 second interval. It would then send a second demand character 60 seconds later. If there was little difference between the data from the two scans, the program would then send the next demand character after a longer interval, say 120 seconds. The program would continue incrementing the time between sending the demand character until some minor differences were observed between scans, suggesting that the workload has begun to change.

Similarly, if two successive scans showed large differences, the implication is that the workload has changed significantly between these scans. In this case, the analysis program might reduce the time between sending the demand character until two successive scans showed similar results.

This variable scan interval is a very important feature of DSTAT, since it allows the host based analysis program to adapt to changing workloads. If the workload varies slowly, scans intervals will be long. If there is suddenly a large change in the output data, the

analysis program can reduce the scan interval until such time as two successive scans show similar results, thereby effectively "tracking" changes in the workload. This type of behavior is important if a host based tuning program is to be developed.

Additional Possibilities

With the amount of data collected by DSTAT, there is an opportunity for the development of other host based programs. A brief description of two of the more interesting ones follow.

Workload Analysis

In order to perform accurate "what if" analysis, a comprehensive picture of the I/O workload is required. With knowledge of the I/O stream, modeling and simulation programs may change the I/O subsystem configuration, apply a workload with similar characteristics to the modified system, and examine any performance differences.

Traditionally, I/O workloads have been characterized by capturing all I/O packets, then performing an analysis on the complete collection of data. On a system of any size, it is clear that the large numbers of I/O packets (numbering in the millions per day) make any comprehensive analysis quite time consuming. Additionally, since there is some question as to the validity of conclusions based on individual I/O packets, the need to capture *every* I/O command is doubtful.

DSTAT provides an ideal tool for the characterization of I/O workloads, since it outputs one group of data that provides summary information over a short time period. Indeed, multiple data collections by DSTAT is quite similar to the method of batch means, a statistical methodology that provides very accurate information about the characteristics of the workload.

With the amount of derived data available with minimal calculations from the DSTAT output, the workload collection effort would also provide large amounts of information for modeling and simulation of other I/O dependent features, such as host based caching, adapter simulations, etc.

Auto Tuning

Perhaps the greatest benefit of DSTAT is when it is used in conjunction with a host based auto tuning application. In this environment, the host based driver will run DSTAT and collect data from one pass. It will then analyze the information (as suggested in the previous section on performance analysis). Once conclusions such as modifying read cache thresholds have been reached, the host based program will connect to the HSx and send the appropriate CLI commands to effect the modifications.

Following this, the host based program will evaluate the next DSTAT data output to see if the changes made were beneficial. If they were not, the host based program could "back out" the changes, or even slightly reduce the amount of change by issuing more CLI commands.

In this manner, the host based program would gradually tune the individual HSx performance parameters until such time as the weighted sum of all response times was minimized. As workload conditions changed, the DSTAT output would reflect any decrease in performance, allowing the host based program to once again begin modifying the HSx parameters in an attempt to maintain peak performance.

Revision History

September 19, 1996:	Initial release.
October 18, 1996:	Demand output character changed to Ctrl/W.
	Documented bug in demand data output that results in [EOP] and [EOD] indicators not printing.
	Documented bug in demand data output that does not ouput return character (hex 0D) on completion of last data line.
	Documented bug in output that does not print a return character (hex 0D) on completion of [EOD] indicator.
May 1, 1998:	Documented the bug that sometimes causes thewrite commands to decrement.