NAME

statistics – Shore performance information

SYNOPSIS

```
pstats
cstats
```

DESCRIPTION

The above commands are commands to the Shore Value-Added Server process's Tcl shell (terminal interface). The command **pstats** prints a plethora of server statistics. The command **cstats** clears the statistics that are collected for the entire server. Some statistics are collected for each client; those are not cleared, and the per-client statistics disappear when the client disconnects from the server. The statistics are described below.

The Shore Value-Added Server also keeps myriad statistics in the client process (library), which are also described here.

All the descriptions below include a name, which is also a manifest constant that is available to Shore applications that use the statistics-gathering programming interface (see **statistics(fc)**).

CLIENT STATISTICS

Batching statistics for TCP

This module describes the client library's batching of update requests into single remote operations (RPCs). These statistics are subject to change, as the interaction between the two processes becomes more sophisticated and more parallelism is introduced in future releases.

BATCH__hit_qmax
: The number of times the batch queue was flushed because it reached its capacity (in number of requests).

BATCH__hit_tcpmax
: The number of times the batch queue was flushed because the socket buffer capacity was reached either locally or at the server.

BATCH__sent
: The number of times the batch queue was flushed because the user requested it.

BATCH__batches
: The total number of batches flushed (sent).

BATCH__avgsent
: The average number of requests per batch.

BATCH__queued
: The total number of requests batched.

BATCH__min_capacity
: The difference between the BATCH__max_capacity (below) and the maximum number of bytes queued in the socket buffer.

BATCH__max_q_len
: The capacity of the queue in number of requests.

BATCH__max_capacity
: The capacity of the queue on number of bytes. This is the smaller of the sizes of the local socket buffer and the server's socket buffer.

The following module describes shared-memory activity. This applies only when the client process and the server are running on the same machine, and when shared memory has been allocated for communication between the two processes (it is by default; you can turn it off with options.) (Using shared memory does not eliminate all use of TCP.)

SHMBATCH__hit_shmmax
: The number of times the batch queue was flushed because there was no more shared memory for storing requests. **options(svas)** for information about allocating shared memory for updates.

SHMBATCH__whole
: Total shared memory allocated for batching requests.

SHMBATCH__min_left
: The difference between the SHMBATCH__whole and and the maximum number of bytes queued in shared memory.

SERVER STATISTICS

The Shore threads package maintains the following statistics. Many of them are of interest only to those people who are writing value-added servers, or to the Shore developers. Many of these statistics will be removed in later releases of Shore.

STHREAD_namemallocs
: The number of times the package invoked a **fast malloc** for named threads. (Threads are named to aid debugging.)

STHREAD_ctxsw
: Context switches between threads.

STHREAD_spins
: Times the process performed a busy-wait on a spin-lock (it contended with a ‘diskrw’ process for a mutual-exclusion variable).

STHREAD_io
: Calls to perform disk I/O through a ‘diskrw’ process.

STHREAD_ccio
: Concurrent calls to perform disk I/O.

STHREAD_iowaits
: Times the server process waited for a ‘diskrw’ process to deliver its goods.

STHREAD_selects
: Total number of Unix select() calls.

STHREAD_selfound
: Times select returned before its timeout, because a file descriptor was ready for input or output.

STHREAD_eintrs
: Times select was interrupted by a signal (normally from the ‘diskrw’ process).

STHREAD_idle
: Times select() returned because its timeout expired.

STHREAD_zero
: Times the ‘diskrw’ started and finished an operation before the server called select(). (The server was busy while the I/O was being performed.)

STHREAD_one
: Times the server called select() once between the started and end of an I/O operation.

STHREAD_two
: Times the server called select() twice between the started and end of an I/O operation.

STHREAD_three
: Times the server called select() three times between the started and end of an I/O operation.

STHREAD_more
Times the server called select() more than three times between the started and end of an I/O operation.

STHREAD_wrapped
Times the select counter wrapped.

The following statistics are kept by the diskrw process (which you never see) and by the server process. A value-added-server writer might modify the ‘diskrw’ process to print its statistics for serious debugging, but in general, the server’s versions of these statistics are adequate. (The descriptions below are from the server’s point of view; the meanings for the ‘diskrw’ process are somewhat different.) The server process notifies the ‘diskrw’ process that an I/O request is queued by sending a SIGUSR1 signal. The ‘diskrw’ process notifies the server that the I/O request is satisfied by sending a SIGUSR2 signal. Both processes use ALRM signals to poll their queues every once-in-a-while if necessary.

DISKRW_alarm
ALRM signals received.

DISKRW_notify
SIGUSR1 sent to ‘diskrw’ process.

DISKRW_kicks
SIGUSR2 received.

DISKRW_falarm
Server found a message in a queue because of an ALRM signal.

DISKRW_fnotify
Server found a message in a queue because of a signal from the other process (SIGUSR2).

DISKRW_found
Server found a message by polling the queue at any other time.

DISKRW_lastsig
The signal number of the last signal received from the ‘diskrw’ process.

DISKRW_spins
Not used by the server.

The Shore Storage Manager maintains a large set of statistics described in **statistics(ssm).**

The RPC service keeps the following statistics:

SVC_STAT_replies_success
RPC replies indicating success.

SVC_STAT_replies_noproc
RPC replies indicating "no such procedure".

SVC_STAT_replies_noprog
RPC replies indicating "no such program".

SVC_STAT_replies_progmismatch
RPC replies indicating "version mismatch".

SVC_STAT_replies_nodecode
RPC replies indicating "couldn’t decode request".

SVC_STAT_replies_systemerr
RPC replies indicating "system error".

SVC_STAT_replies_denied
RPC replies indicating "denied" -- authentication failure.

TCPSVC_STAT_replymax
: Largest reply sent over TCP.

TCPSVC_STAT_reqmax
: Largest request received over TCP.

These are internal to the UDP RPC service and are subject to change:

UDPSVC_STAT_cache_finds

UDPSVC_STAT_cache_nofinds

UDPSVC_STAT_cache_hits

UDPSVC_STAT_cache_misses

UDPSVC_STAT_cache_sets

UDPSVC_STAT_cache_presets

UDPSVC_STAT_cache_gets

UDPSVC_STAT_recvfroms

These might be of interest to the user; they apply only to the UDP services (MOUNTD and NFSD).

UDPSVC_STAT_drops
: Number of requests dropped because for lack of buffering capacity.

UDPSVC_STAT_done
: Requests serviced.

UDPSVC_STAT_replies
: Replies sent.

UDPSVC_STAT_rereplies
: Re-transmitted replies.

UDPSVC_STAT_retrans
: Number of different requests received having retransmissions.

UDPSVC_STAT_retrans_total
: Total number of retransmitted requests received (for any requests).

UDPSVC_STAT_retrans_max
: Maximum number of retransmissions detected for any given request.

UDPSVC_STAT_replymax
: Largest reply sent over UDP.

UDPSVC_STAT_reqmax
: Largest request received over UDP.

These statistics are kept for the NFS service:

NFSD_lock_timeouts
: The number of times an operation could not be performed because it would have to wait for lock.

NFSD_nosuch_errors
: The number of errors for which there is no reasonable NFS error response.

The module called "System properties cache (per-client)" will be removed in the next release.

The Shore value-added-server contains four modules that serve different RPC interfaces: NFS, MOUNT (for NFS), clients, and other Shore value-added servers. For each of these services, there is a module that counts the numbers of RPC requests received.

STATISTICS ON SERVER AND CLIENT

Both processes count the requests received. In the client library, two sets of counts are kept: one to reflect the local requests ("Client Requests"), and another to reflect the remote operations, or requests forwarded to the server ("Messages Sent"). These two sets of numbers may differ because sometimes a single local request is broken into several remote requests. For example, a request to create a very large object with initialized data may result in a request to create an object followed by a series of write or append requests.

On the server, they reflect the remote operations requested by the client ("Client RPCs"), which might be less than the client's count because some can be satisfied locally on the client. Shore users are discouraged from using the constants to print individual statistics in this group because the constants are generated by the RPC package, and they change with each change to the client-server protocol. For that reason, we do not make the constants manifest. Nevertheless, if you print the statistics from the Shore server's terminal interface, or with the object cache *oc_pstats* option, you will see these counts.

The Unix resource usage statistics are collected and reported in both processes. The defined constants for these statistics are:

UNIX_utime_tv_sec

UNIX_utime_tv_usec

UNIX_stime_tv_sec

UNIX_stime_tv_usec

UNIX_ru_idrss

UNIX_ru_minflt

UNIX_ru_majflt

UNIX_ru_nswap

UNIX_ru_inblock

UNIX_ru_oublock

UNIX_ru_msgsnd

UNIX_ru_msgrcv

UNIX_ru_nsignals

UNIX_ru_nvcsw

UNIX_ru_nivcsw

VERSION

This manual page applies to Version 1.1 of the Shore software.

SPONSORSHIP

The Shore project is sponsored by the Advanced Research Project Agency, ARPA order number 018 (formerly 8230), monitored by the U.S. Army Research Laboratory under contract DAAB07-91-C-Q518.

SEE ALSO

statistics(ssm), statistics(oc), and **statistics(fc)**