```
            "available" : <num>,
            "totalTickets" : <num>
         },
         "read" : {
            "out" : <num>,
            "available" : <num>,
            "totalTickets" : <num>
         }
   }
},
```

The final `ok` field holds the return status for the `serverStatus` command:

```
"ok" : 1
```

5.3.5 Journaling Mechanics

When running with journaling, MongoDB stores and applies *write operations* (page 73) in memory and in the on-disk journal before the changes are present in the data files on disk. Writes to the journal are atomic, ensuring the consistency of the on-disk journal files. This document discusses the implementation and mechanics of journaling in MongoDB systems. See *Manage Journaling* (page 233) for information on configuring, tuning, and managing journaling.

Journal Files

With journaling enabled, MongoDB creates a journal subdirectory within the directory defined by `dbPath`, which is `/data/db` by default. The journal directory holds journal files, which contain write-ahead redo logs. The directory also holds a last-sequence-number file. A clean shutdown removes all the files in the journal directory. A dirty shutdown (crash) leaves files in the journal directory; these are used to automatically recover the database to a consistent state when the mongod process is restarted.

Journal files are append-only files and have file names prefixed with `j._`. When a journal file holds 1 gigabyte of data, MongoDB creates a new journal file. Once MongoDB applies all the write operations in a particular journal file to the database data files, it deletes the file, as it is no longer needed for recovery purposes. Unless you write *many* bytes of data per second, the journal directory should contain only two or three journal files.

You can use the `storage.smallFiles` run time option when starting `mongod` to limit the size of each journal file to 128 megabytes, if you prefer.

To speed the frequent sequential writes that occur to the current journal file, you can ensure that the journal directory is on a different filesystem from the database data files.

Important: If you place the journal on a different filesystem from your data files you *cannot* use a filesystem snapshot alone to capture valid backups of a `dbPath` directory. In this case, use `fsyncLock()` to ensure that database files are consistent before the snapshot and `fsyncUnlock()` once the snapshot is complete.

Note: Depending on your filesystem, you might experience a preallocation lag the first time you start a `mongod` instance with journaling enabled.

MongoDB may preallocate journal files if the `mongod` process determines that it is more efficient to preallocate journal files than create new journal files as needed. The amount of time required to pre-allocate lag might last several minutes, during which you will not be able to connect to the database. This is a one-time preallocation and does not occur with future invocations.

To avoid preallocation lag, see *Avoid Preallocation Lag* (page 234).