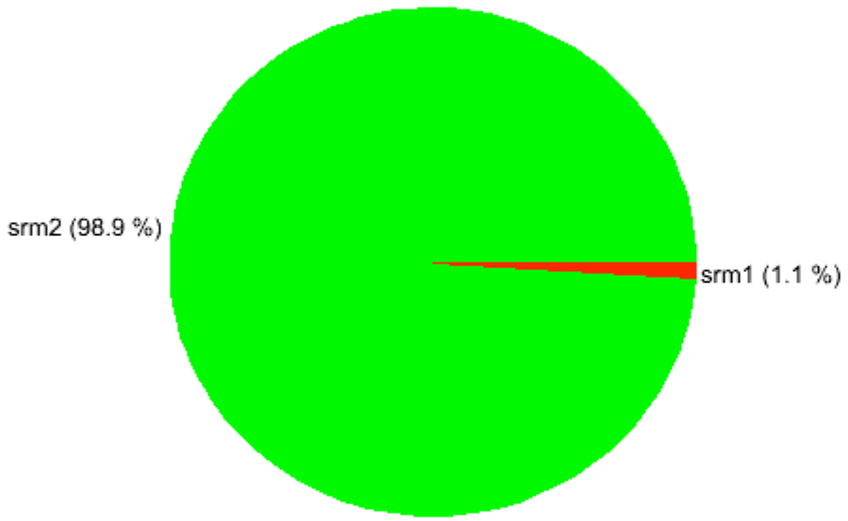
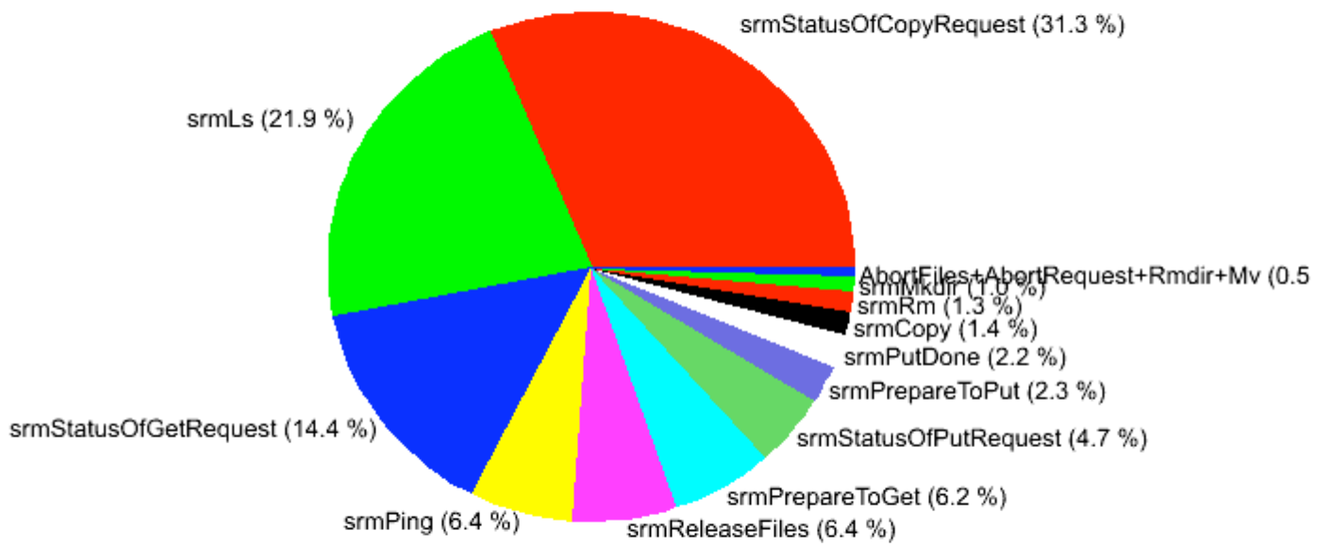
Protocol obstacles to SRM performance

Matt Crawford
Fermilab

Table of Contents

Number of Requests





srmPrepareToGet ≃ 6.2%

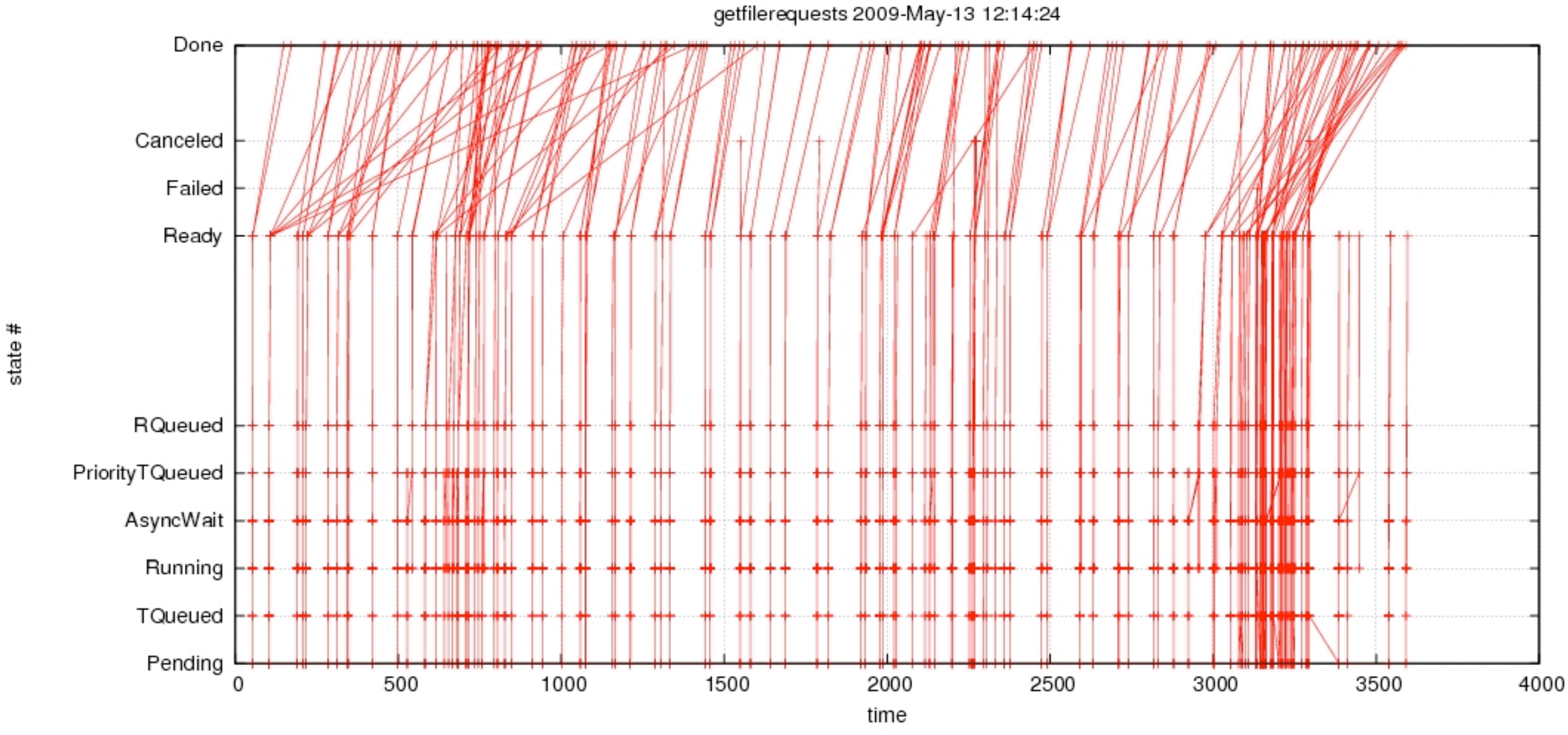
srmPrepareToPut ≃ 2.3%

srmLs ≃ 21.9%

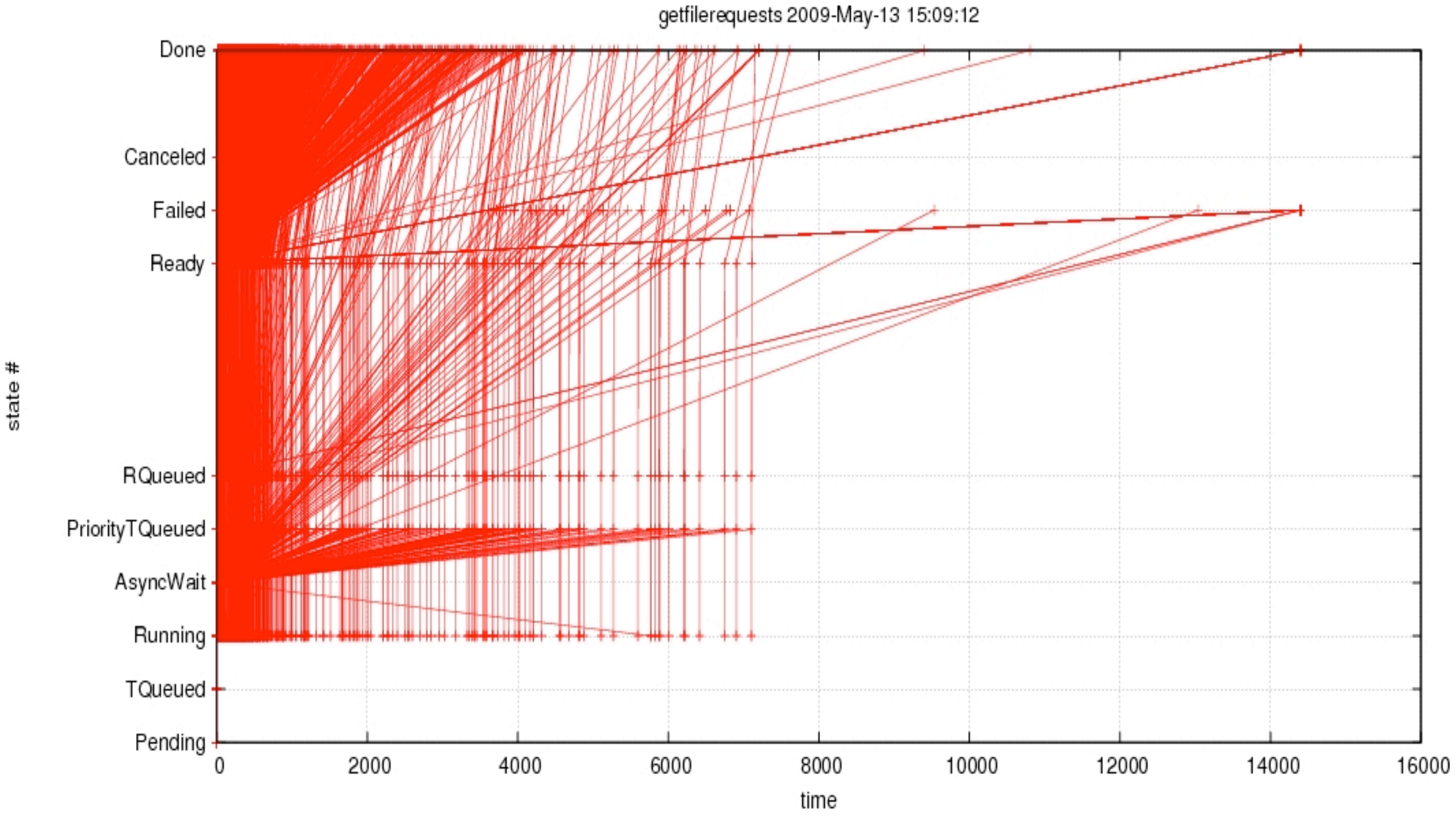
∑(status queries) ≃ 50.4%

srmCopy only 1.4%, but each call is for many files

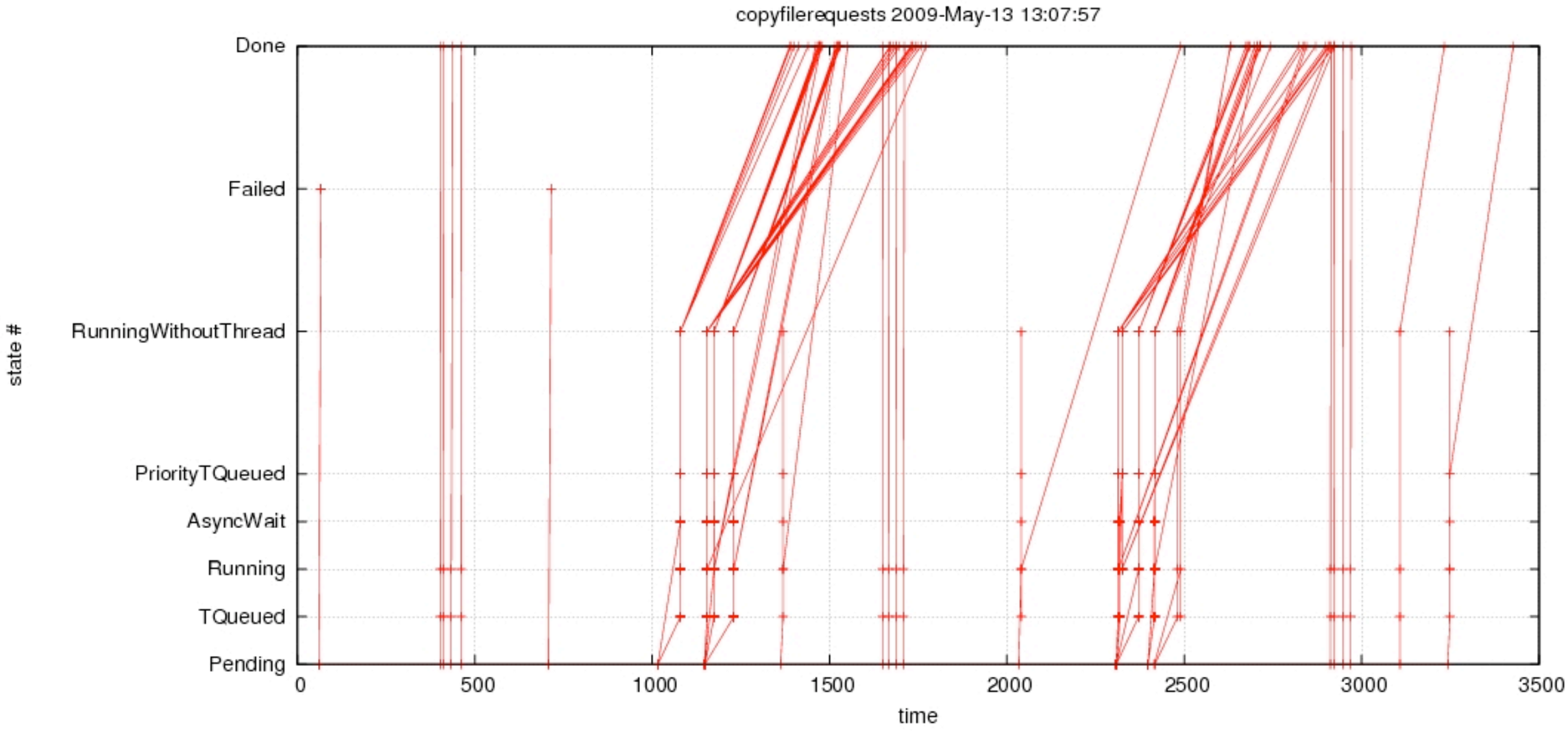
“Get” states–Absolute time



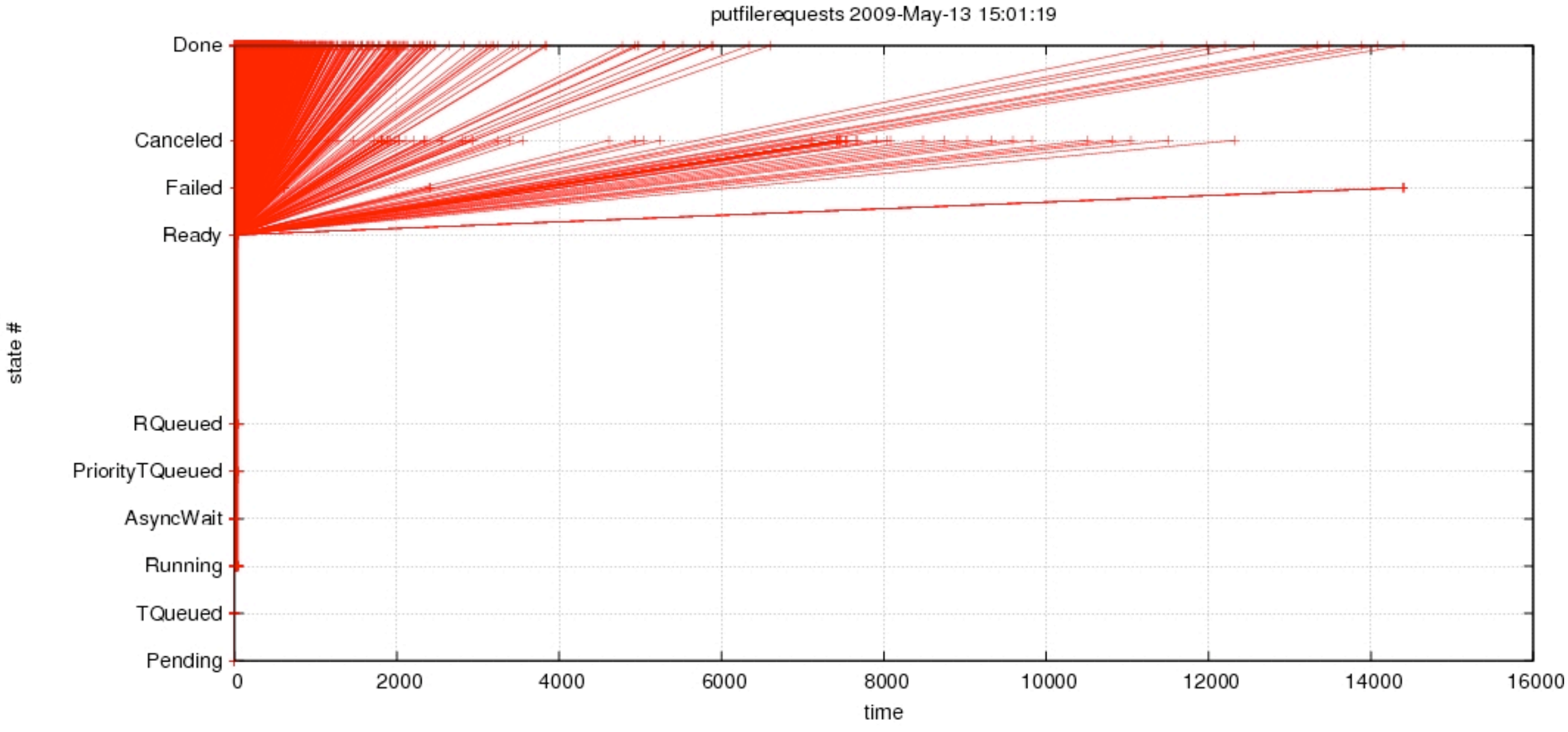
“Get” states–Relative time



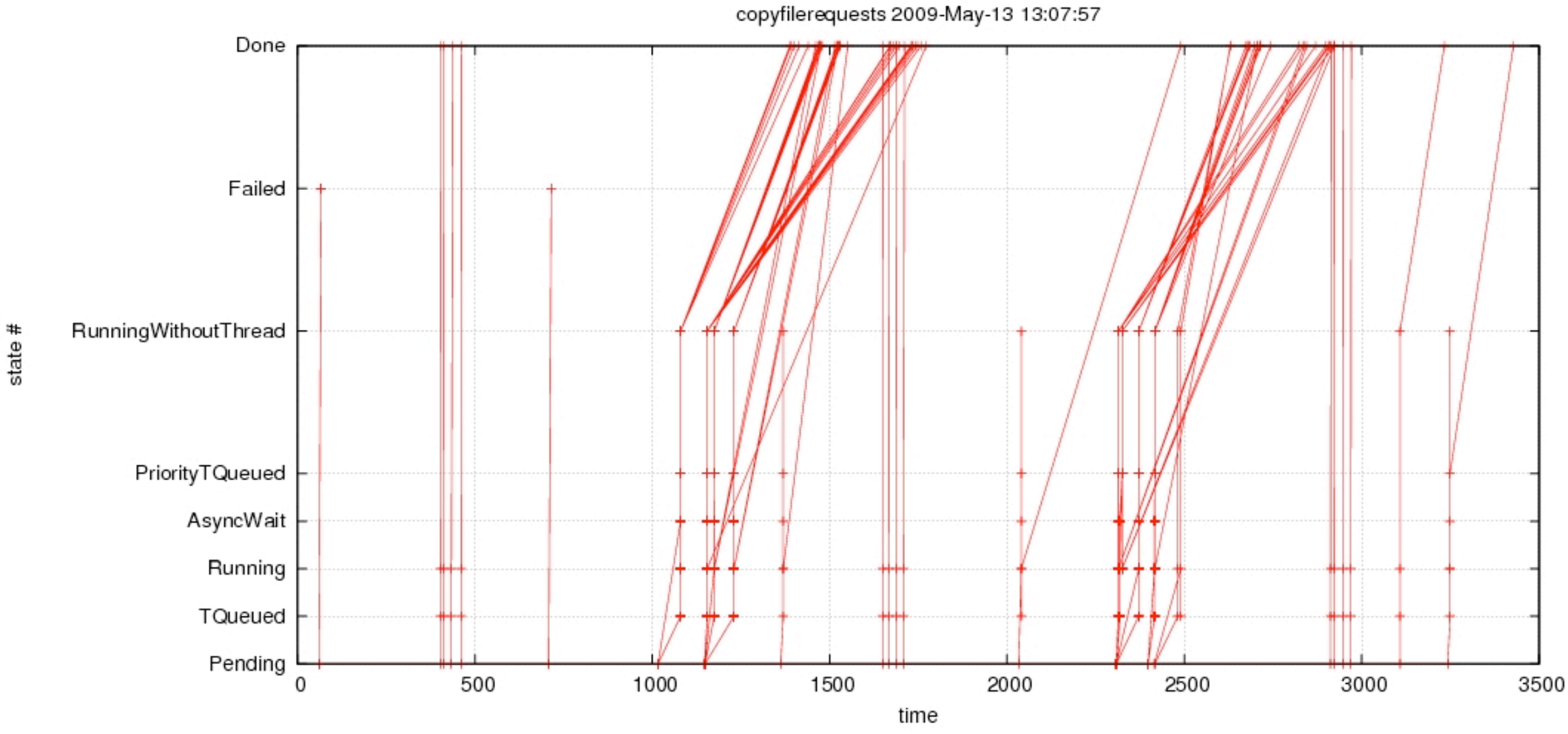
“Put” states–Absolute time



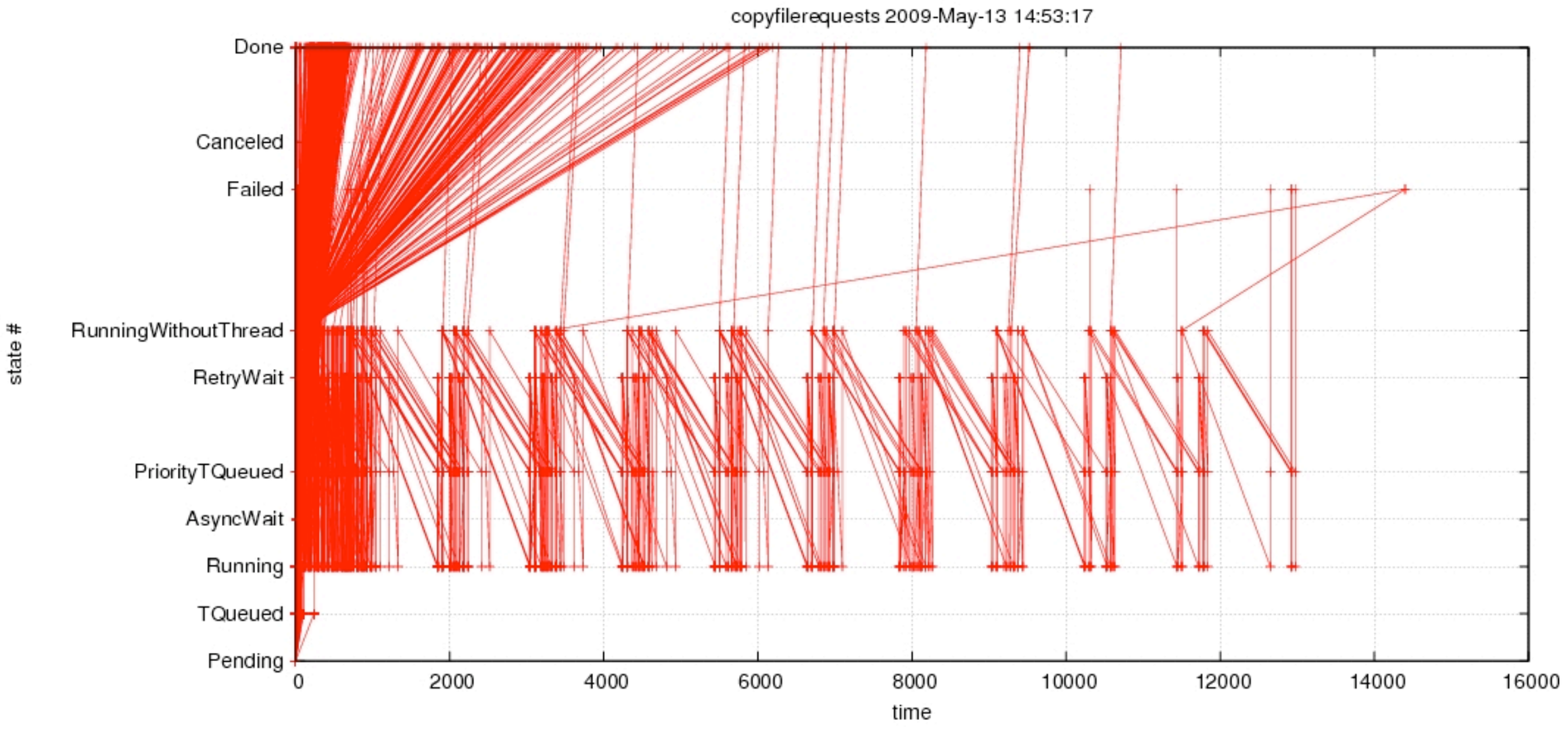
“Put” states–Relative time



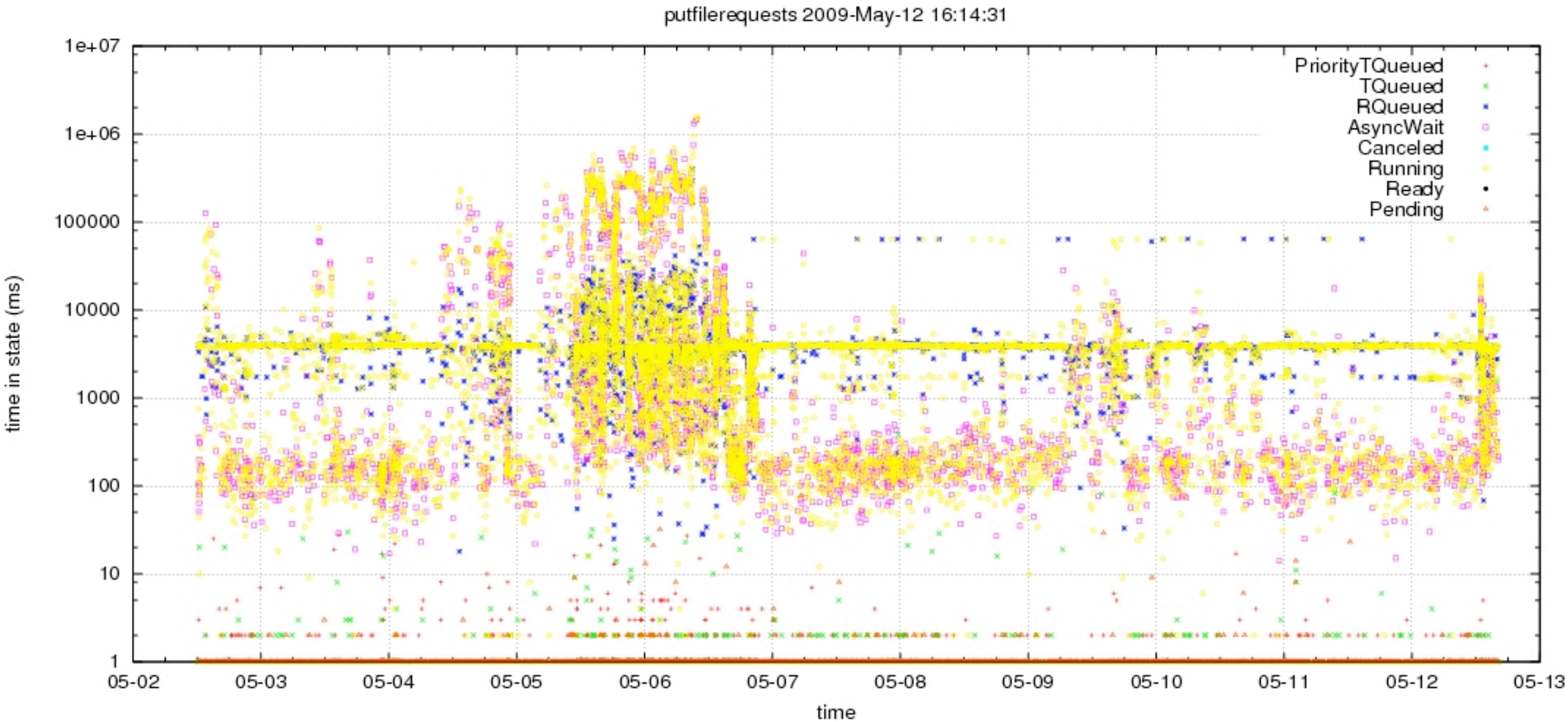
“Copy” states–Absolute time



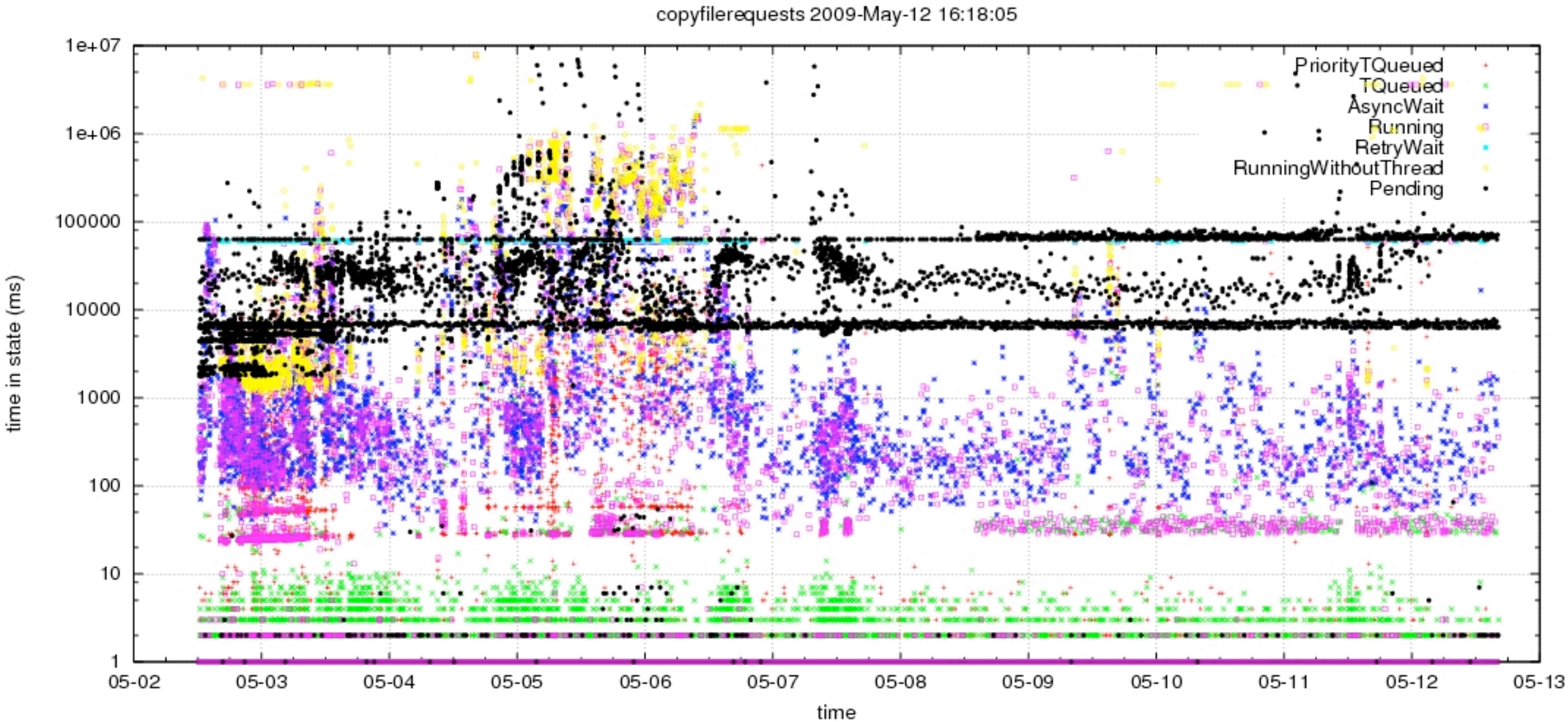
“Copy” states–Relative time



Time in states–"Put"



Time in states–"Copy"



Strategic Concerns

A storage element behind an SRM serves many competing users and groups.

- Must protect itself from wasteful requests.
- Must protect users (⇒ itself) even from orderly requests.

SE protects itself from SRM, or trusts SRM to protect from users?

Giving TURLs to clients is one means of regulating work.

Protocol obstacles

Clients have too much control!

* Credential delegation concentrates computational work $O(n^4)$ on the server.
* Bad decision to not support SRM BUSY in clients.
* Questionable design to have clients poll to get status.

Protocol obstacles

- Incremental deployment of better methods is not easy. Clients do not fetch WSDL(?) and there is no exchange of supported optional feature lists.

Offloading SRM

- WS-Delegation to accept delegated credentials.
- Have server notify clients of ready or completed results: fewer authentications (on average), less time holding resources.
  - Clients behind NAT or firewall? WS-Notification.

All of these involve significant protocol changes.

- TLS with session restart?

Deferring work

Deferring or refusing work is an effective strategy *up to a point*. Recent request service times may be input to a control loop.

After that point, the cost imposed on other parts of the aggregate data-handling and data-processing system must be considered.

Deferring access (for hours?) by computational elements may raise objections.

SRM may wish to opt out of that function.

Specification Precision

Enumerate the visible (but possibly abstract) state-variables of the storage system.

- Divide sharply from the implementation-dependent states of the underlying storage!

Sharply define the required preconditions and effects of each client function in terms of the visible states.