PROOF – Parallel ROOT Facility

B. Bellenot, R. Brun, G. Ganis, J. Iwaszkiewicz, F. Rademakers, *CERN, Geneva, Switzerland*,
M. Ballintijn, *MIT, Cambridge, MA, USA*

PROOF

PROOF enables interactive analysis with ROOT [1] on distributed computing resources. It realizes basic parallelism by exploiting the independence of uncorrelated events.

PROOF is designed for use at
- **Central Analysis Facilities**
- **Departmental workgroup computing facilities (Tier-2's)**
- **Multi-core, multi-disks desktops**

Design goals

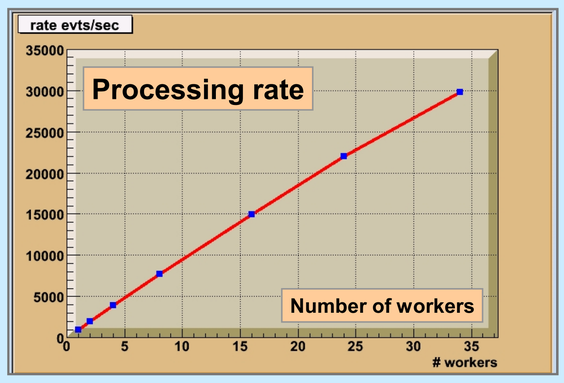
- **Transparency: distributed system perceived as an extension of the local ROOT session (same: syntax, scripts, …)**
- **Scalability: efficient use of the available resources: performance scales with the number of CPUs and disks.**
- **Adaptability: adapt to heterogeneous resources**

Features

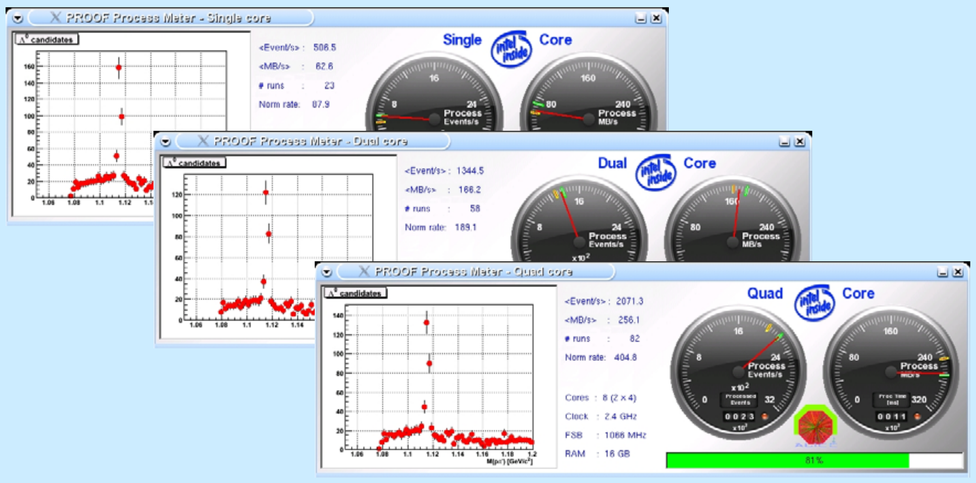
- **Package manager**
  - Handling of additional libraries, classes, data files, etc.
- **Support for dynamic environment setting**
  - On-the-fly definition of variables and/or sourcing of relevant scripts
- **Query manager for easy handling of results**
  - Results can be saved on any mass storage
- **Support for "interactive batch"**
  - **Smooth interactive -> batch transition**
  - **Client disconnection / reconnection**
    - Users can reconnect later from a different place to e.g. check a long-query status and retrieve the results
  - **Background, non-blocking running model**
  - **Multiple-session control from single ROOT shell**
    - Concurrent execution of queries on different sessions
- **Dataset manager and uploader**
  - Handling of meta information about data sets
  - Browsing functionality
- **Real-time feedback**
  - Snapshots of selected objects received at tunable frequency
- **User runs in a unique sandbox**
  - Flexible authentication framework (password, Krb5, GSI)

Scalability

- **ALICE and PROOF Central Analysis Facility (CAF)**
- **34 Xeon 2.8 GHz**
- **4 GB RAM, GB Ethernet**
- **Data read from mass storage via XROOTD [2]**
- **Processing rate up to 700MB/s**



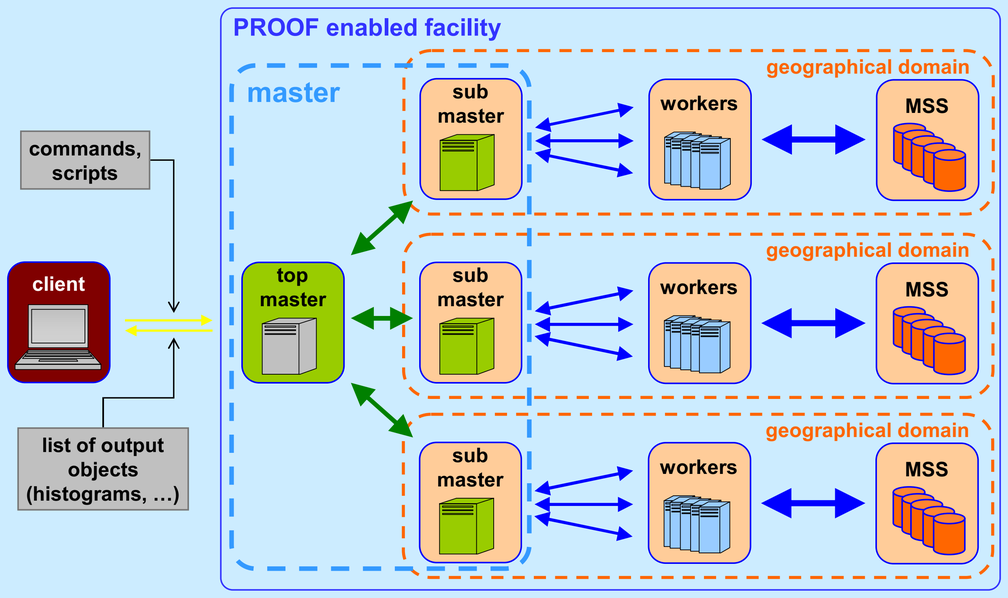
Multicore Scalability



- **ALICE analysis repeated *ad infinitum* on dual-socket machines equipped with quad-, dual- and single-core processors**
- **Speed-o-meters show the instantaneous event and MB processing rates: the advantage of having more CPU is clear**
- **The rate normalized by the clock speed and # of CPU sockets scales nearly with the # of cores, indicating that the available computing power is fully exploited**

[1] http://root.cern.ch
[2] http://www.slac.stanford.edu/xrootd

Multi-tier architecture



Connection lay-out set up via dedicated daemons in charge of authenticating the clients and spawning the server applications. XROOTD [2] has been instrumented for this purpose.

Work distribution and Data Access Strategies

Low-latency access to data is crucial
- **Optimizing to process data in it's current location**
- **Using caching and pre-fetching techniques for data from mass storages**

Dynamic load balancing: pull architecture
- **Workers ask for more work when they are ready**
- **Packet generation (*packetizer*):**
  - **Adaptive mechanism to avoid node overloading**
  - **Packet size calculation based on a fixed time quantum**

Resource Scheduling

To face the needs of large, multi-user analysis environments expected in the LHC era, optimized sharing of resources among users is required.
The resource scheduling improves the system utilization, insures efficient operation with any number of users and realizes the experiment's scheduling policy. To achieve the goal, two levels of resource scheduling are introduced:
- **At worker level, a dedicated mechanism controls the fraction of resources used by each query, according to the user priority and current load.**
- **At master level, a new central component, the *scheduler,* decides which resources can be used by a given query, based on the overall status of the cluster, the query requirements (data location, estimated time for completion, …), the client history, etc.**

Running PROOF: ROOT shell or GUI

A PROOF session is controlled by a dedicated class which can be instantiated on the ROOT shells or within ROOT-enabled applications (see the *grid interface* box for an example).
A full-featured graphical controller is also available.