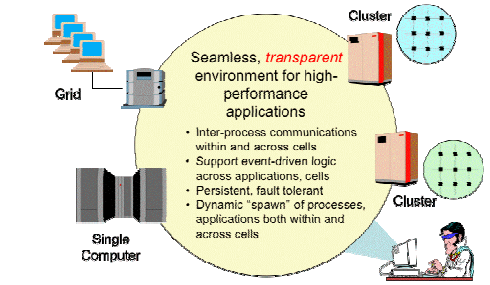
Open Run-Time Environment

Jeff Squyres

ORTE

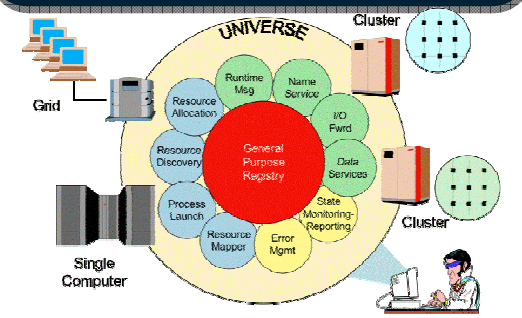
- Run-time support system
  - Basis for Open MPI launch, kill, etc.
  - But can be used independently
- Ties into back-end run-time environments
  - …or not!
- Started as tiny subsystem in OMPI
  - Evolving into its own project
  - Other projects using ORTE without OMPI
  - May [someday] be a separate project

ORTE Objectives



*Cell = one or more computers sharing a common launch environment/point

ORTE Architecture



The ORTE *Universe*

- Collection of services and resources
  - Supports multiple simultaneous applications
  - Configurable environment
  - Maintains system status, inter-process coordination
  - Monitors state-of-health
    - Processes, resources

The ORTE *Universe*

- Head Node Process (HNP)
  - Resides on machine from which processes are launched on that cell
    - E.g., front end of a cluster, grid master
  - Responsible for…
    - Launching all processes on that cell
    - Monitoring cell state-of-health (nodes, processes)
    - Reporting cell state to rest of universe
    - Routing communications between cells

Uniqueness

- User can have multiple simultaneous universes
  - Named or "default"
- Jobs and processes within a given universe can communicate, synchronize
- Access restrictions
  - Scope can be set by user
    - Public - accessible by anyone (be careful!)
    - Private - accessible by user only (default)
    - Exclusive - dedicated to a specific job, no subsequent connections allowed
  - Relies on operating system security

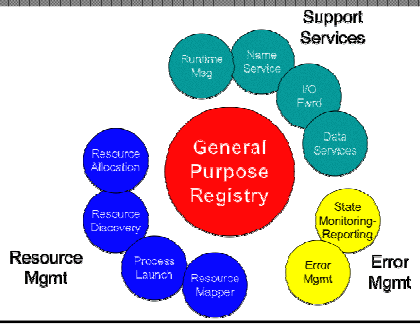
Universe Types

- Non-persistent universe (current default)
  - Ends with application completion
- Persistent universe
  - Exists outside of any particular application
  - Used for multiple synchronized application operations across cells
  - In MPI context, frequently used for MPI-2 dynamic operations
    - Connect, accept, join

Universe Globals

- Process name: <jobid>.<pset>.<vpid>
- Job (jobid)
  - Unique within a given universe
  - One issued per each execution of "orterun"
    - Note: "orterun" = "mpiexec" = "mpirun"
- Process set (pset)
  - Collection of processes within a given job that were initiated with a common "spawn"
  - Unique within a given *job*
  - pset=0 reserved for daemons that might be launched by the job
- Virtual process ID (vpid)
  - ID of process - unique within a given *process set*
  - Usually equal to the MPI_COMM_WORLD rank

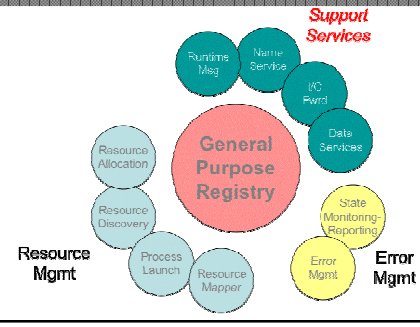
Universe Elements



General Purpose Registry

- Data storage/retrieval system
  - All common data types plus user-defined
  - Heterogeneity between storing process and recipient automatically resolved
  - Still a single instance; working on distributed
- Publish / subscribe
  - Support event-driven coordination and notification
  - Subscribe to individual data elements, groups of elements, wildcard collections
  - Specify actions that trigger notifications, information to be returned

Universe Elements



Runtime Messaging Layer

- Single point-of-contact for routing and delivery of messages within ORTE
  - Not intended for high-performance, large message communications
  - Inter-cell routing
  - Inter-universe messaging *not* supported
- Guaranteed delivery
  - Blocking, non-blocking
  - Broadcast, process-to-process
- Multiple parallel network transports
  - Out-of-band (OOB) framework auto-selects available transports
  - RML selects "best" option(s)
  - Message fragmentation *not* supported
  - Auto-update of connection information to support addition, deletion of processes
- Heterogeneity automatically resolved
  - Byte order, size differences

Data Services

- Single interface for all declared data types
  - Register data types, manipulation functions
  - Unstructured or structured
- Pack / unpack for network communications
  - Resolve data heterogeneity issues
  - Construct / deconstruct buffers for transmission over RML
- Support transparent data manipulation within ORTE
  - All declared data types
    - Copy, compare, size, print, release
  - Arithmetic functions for integer data types
    - Add, subtract, divide, multiply
    - Increment, decrement

Name Services & I/O Forwarding

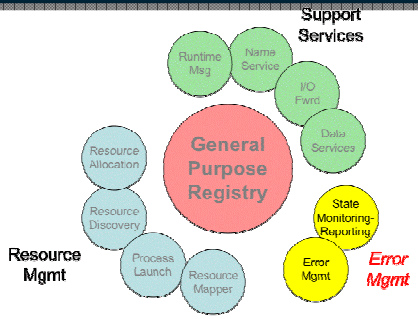
Name Services

- Generate unique names
- Support name passing to child processes
- Provide support functions
  - Get peers for process sets, jobs

I/O Forwarding

- Source / sink: file
  - (including stdin / out / err)
- From application start
  - Setup before main()
- Only basic usage currently supported by mpirun
  - We should do more!

Universe Elements



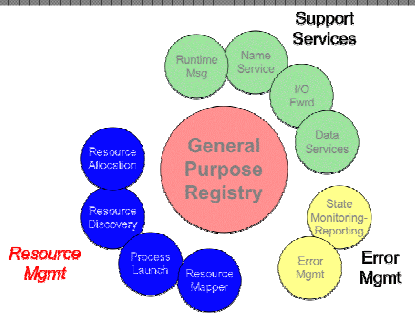
State Monitoring & Reporting

- Single point for reporting changes in state
  - Report changes in state as detected by system
  - Notification to all interested subsystems through registry subscription service
- Internal monitoring capabilities
  - Used where system doesn't provide own capability or to augment available services
  - Process state
    - Tracks process successful startup/shutdown, abnormal terminations
  - System state
    - Tracks status, performance
      - Node/cell (up, down, booting, …)
      - Communications (bandwidth, connectivity)
    - Develops model of anticipated performance, fault prediction

Error Manager

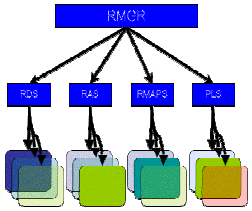
- Log ORTE errors for reporting, future analysis
- Primary responsibility: fault response
  - Contains defined response for given types of faults
  - Responds to faults by shifting resources, processes
- Secondary responsibility: resilience strategy
  - Continuously update and define possible response options
  - Utilizes SMR fault prediction to trigger pre-emptive action
- Allows selection of various response strategies via component system
  - Run-time decision
  - Selectable by command-line option, environmental parameter, or default to local system configuration

Universe Elements



Resource Manager (RMGR)

- Integrated, single point-of-contact for launching jobs, processes
- Selectable components allow multiple strategies for interweaving functional blocks
  - URM component seems to meet nearly all needs
  - Proxy component allows remote processes to access resources on this/other cells, without transferring data



Resource Allocation

- Meant to allocate resources
  - E.g., submit batch job
- Some RAS components currently exist
  - But are really mis-placed
  - Being ported back to RDS (resource discovery)
- Do not have any real RAS components yet
  - Probably only use one component at run-time

Resource Discovery

- Discover what resources have been given to the job
  - In resource manager job (PBS, SLURM, etc.)
  - Hostfile
  - Localhost only
- Supports
  - Hostnames
  - Max process counts on each (slots)
- Use all available components at run-time

Resource Mapping

- Given a set of processes
  - Map them to resources
- Only one component: round_robin
  - Node major and slot major ordering
  - May have more here someday
- Use one component at run-time

Process Launch

- Use a back-end system to launch
  - SLURM, PBS, rsh/ssh, …
- Interface supports process kill as well
- Can only use one PLS component per cell

orterun

- Tool for launching processes in universe
  - Can launch MPI and non-MPI apps
  - Sym linked to mpirun and mpiexec
- Supports MPI-2 mpiexec syntax
  - Supports SPMD and MPMD
  - Supports process-unique MCA parameters
  - Can also give a file with all commands / args
  - --host works, --arch does not
- See the man page (mpirun.1)

orterun Scenario

- mpirun –np 4 a.out
  - RMGR is invoked to spawn the job
  - Query RDS and RAS
  - Get a list of resources
  - Invoke RMAPS to map 4 processes to resources
  - Invoke PLS to launch processes
  - Invoke PLS to wait for processes to complete

MPI Startup

- MPI_INIT determines its identity
- Calls back to GPR as rendezvous point
  - Exchange MPI pt2pt connection information
  - Done as a “compound command”
    - Everything exchanged in one transfer per process
- orterun unaware if MPI or non-MPI job

MPI_COMM_SPAWN

- Essentially the same as orterun
  - Invokes rmgr.spawn()
- Rendezvous point is the GPR
  - Hence, MPI process does not have to double as “orterun” role

Adding Support for New RMs

- Typically add two components
  - RDS: query the RM to find resources allocated to the job
  - PLS: use the RM’s native mechanism to launch, monitor, kill
- Example
  - SLURM has RAS (moving to RDS) and PLS components

Other ORTE Tools

- …none yet
- But others are under development / contemplated
  - Console-like application
  - Screen-like application
  - I/O multiplexer
  - Universe ps, kill, etc.

Ongoing Efforts / Future Work

- Remote launch from desktop/notebook
  - Support disconnect/reconnect
  - Remote status reporting
  - Resource discovery, scheduling
- Multi-cell operations
  - Single application spanning multiple cells
  - Multiple applications synchronized and/or sharing data across multiple cells
- Resilient operations
  - Next-generation response to “faults”

Questions?

Backup Slides

ORTE vs. Grid

OpenRTE	Grid
• Full local autonomy, control ▪ Operates at *user* level ▪ User must be able to spawn process on head node of cell ▪ Utilizes multiple communication protocols	All resources must install and support grid systems, protocols Operates at *admin* level
• Component architecture ▪ Allows rapid, easy prototyping and incorporation of new features ◦ Store component in accessible location ◦ Tell system to use it (command line parameter, environmental variable) ▪ Planned incorporation of grid protocols to allow interaction	Build and distribute web services to add new capabilities, must integrate directly into application

ORTE vs. Grid

OpenRTE	Grid
• Full local autonomy, control • Component architecture • Transparent ▪ No application code changes or “glue” programming to move from cluster to multi-cell operations	Application incorporates grid programming or must be linked to grid-specific libraries Customized programming to utilize multiple cells to subdivide applications, synchronize multiple applications