PMIx: A Tutorial

Ralph H. Castain
Intel
Agenda

- Day 1: Server & Scheduler
  - Overview of PMIx
  - Detailed look at Launch
- Day 2: Client, Tools, & Events – Oh My!
  - Event notification
  - PMIx Client functions
  - PMIx Tool support
Terminology

• **Session**
  - Allocation to a specific user

• **Job**
  - What was submitted to the scheduler for allocation and execution
  - Can span multiple sessions

• **Task**
  - Workflow to be executed within an application
  - Multiple jobs can coexist within a given session
  - In MPI terms, a “task” is synonymous with MPI_COMM_WORLD

• **Application**
  - One or more processes executing the same executable
  - Can be a script, typically a binary
  - A single task can be comprised of multiple apps
Day 1: Detail

- Overview
- PMIx Reference Implementation
- Server Initialization
  - Exercise
- Launch Sequence
  - Exercise
Origin: Changing Landscape

Launch time limiting scale

Programming model & runtime proliferation

Legion

Hybrid applications

Model-specific tools

Container technologies

MPI

OpenMP

Apache Spark

R

Docker
Start Someplace!

- Resolve launch scaling
  - Pre-load information known to RM/scheduler
  - Pre-assign communication endpoints
  - Eliminate data exchange during init
  - Orchestrate launch procedure
Traditional Launch Sequence

1. Job Script
2. WLM
3. FS
4. Wait for files & libs
5. WLM
6. RM
7. Spawn Procs
8. Proc
9. Fabric
10. NIC
11. Topo
12. GO
13. Global Xchg
14. Barrier
15. Launch Cmd

Traditional Launch Sequence

Wait for files & libs
Newer Launch Sequence

1. Job Script
2. WLM
3. WLM
4. FS
5. Wait for files & libs
6. RM
7. Spawn Procs
8. Proc
9. Fabric
10. NIC
11. Topo
12. Proxy
13. Proxy
14. Proxy
15. Proxy
16. Global Xchg
17. Barrier
18. GO
PMIx Launch Sequence

*RM daemon, mpirun-daemon, etc.
Three Distinct Entities

• PMIx Standard
  ▪ Defined set of APIs, attribute strings
  ▪ Nothing about implementation

• PMIx Reference Library
  ▪ A full-featured implementation of the Standard
  ▪ Intended to ease adoption

• PMIx Reference RTE
  ▪ Full-featured “shim” to a non-PMIx RM
  ▪ Provides development environment
Where Is It Used?

- Libraries
  - OMPi, MPICH, Intel MPI, HPE-MPI, Spectrum MPI, Fujitsu MPI
  - OSHMEM, SOS, OpenSHMEM, …
- RMs
  - Slurm, Fujitsu, IBM’s JSM, PBSPro (2019), Kubernetes(?)
  - Slurm enhancement (LANL/ECP)
- New use-cases
  - Spark, TensorFlow
  - Debuggers (TotalView, DDT)
  - MPI
    - Re-ordering for load balance (UTK/ECP)
    - Fault management (UTK)
    - On-the-fly session formation/teardown (MPIF)
  - Logging information
  - Containers
    - Singularity, Docker, Amazon
Async event notification

Cross-model notification
- Announce model type, characteristics
- Coordinate resource utilization, programming blocks

Generalized tool support
- Co-launch daemons with job
- Forward stdio channels
- Query job, system info, network traffic, process counters, etc.
- Standardized attachment, launch methods
Sprinkle Some Magic Dust

- Allocation support
  - Dynamically add/remove/loan nodes
  - Register pre-emption acceptance, handshake
- Dynamic process groups
  - Async group construct/destroy
  - Notification of process departure/failure
- File system integration
  - Pre-cache files, specify storage strategies
PMIx-SMS Interactions

PMIx

PMIx Client

PMIx Server

APP

OpenMP

MPI

Orchestration Requests

Responses

System Management Stack

FS

Fabric Mgr

Fabric

NIC

RAS

Job Script

Tool Support
PMIx-SMS Interactions

Container!

OpenMP

APP

MPI

Orchestratıon
Requests

PMIx
Client

PMIx
Server

Responses

PMIx

System
Management Stack

FS

Fabric

Mgr

NIC

RAS

Job
Script

Tool Support
Philosophy

• Generalized APIs
  ▪ Few hard parameters
  ▪ “Info” arrays to pass information, specify directives
• Easily extended
  ▪ Add “keys” instead of modifying API
• Async operations
• Thread safe
Guiding Principles

• Messenger, not a Doer
  ▪ There are some (very limited) exceptions

• No internal inter-node messaging support
  ▪ Per RM request, all inter-node messaging provided by host environment
    • Minimizes connections and avoids yet another wireup procedure
  ▪ Host environment required to know where things are
    • Where to send requests based on PMIx server type, info on a given proc

• “Not Supported”
  ▪ Critical to RM adoption
  ▪ Let the market drive support
“Doer” Exceptions

• Interactions with non-PMIx systems
  ▪ Fabric manager, credential subsystems, storage systems

• Aggregate local collective operations
  ▪ Fence, connect/disconnect

• Environment “support”
  ▪ Inventory collection, process monitoring, logging
PMIx Scope

- **Wireup**
  - Fence, put, get, commit
- **Publication**
  - Publish, lookup, unpublish
- **Dynamics**
  - Spawn, connect, disconnect, group construct/destruct
- **Storage**
  - Estimate retrieval times, set hot/warm/cold policy, data movement
- **WLM**
  - Inventory, comm costs, subsystem app resource allocations, allocation mgmt
- **Fabric**
  - QoS control, async updates
- **Tools**
  - Query, attach/detach, IO fwd
- **Events (Async notification)**
- **Info**
  - Query, logging
WLM/RTE → Orchestrator

- Network
- File System
- Monitoring
- DB
- WLM
- APP
- Console
- Prov. Agent
Day 1: Detail

- Overview
- PMIx Reference Implementation
- Server Initialization
  - Exercise
- Launch Sequence
  - Exercise
Objective
- Ease adoption, validate proposed standard modifications/additions

Written in C with some C++ like extensions (object classes)

Plugin architecture
- Internal APIs defined as “frameworks” with individual ”component” implementations
- Components loaded as dll’s to allow for proprietary add-ons

Python bindings
- Utilize public PMIx APIs (not internal)

Debugging fundamentals - Verbosity is your friend
- Framework level spans components (e.g., pml_base_verbose)
  - No separation between client and server
- Functional level (pmix_iof_xxx_VERBOSE), where xxx is either “client” or “server”
Releases

RM Production Releases

2014

1/2016
1.1.3
Launch & wireup

12/2016
1.2

12/2017
2.0
Events, fabric, & basic tool

12/2018
3.0
Logging, IO fwd, credentials, inventory, job ctrl, monitoring, dyn alloc

12/2019
4.0
Scheduler, groups, storage, adv tools, Python

https://github.com/pmix/pmix/releases
Cross-Version Support

- Auto-negotiate messaging protocol
- Client starts
  - Envar indicates server capabilities
  - Select highest support in common
  - Convey selection in connection handshake
- Server follows client’s lead
  - Per-client messaging protocol
  - Support mix of client versions

<table>
<thead>
<tr>
<th>PMIx version</th>
<th>Server &gt;= Client</th>
<th>Any client/server combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMIx v1.1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v1.2.5+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v2.0.3+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v2.1.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v3.0.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v3.1.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMIx v4.x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Done!
**Process Types**

- **Client**
  - Application process connected to local server

- **Server**
  - Client + server APIs + host function module
  - Subtypes: gateway, scheduler, default

- **Tool**
  - Client APIs with rendezvous

- **Launcher**
  - Tool + server APIs
Day 1: Detail

- Overview
- PMIx Reference Implementation
- Server Initialization
  - Exercise
- Launch Sequence
  - Exercise
Server Initialization

- Declare server type
  - Gateway: acts as a gateway for PMIx requests that cannot be serviced on backend nodes (e.g., logging to email)
  - Scheduler: supports inventory and application resource allocations
  - Default: supports local PMIx clients and possibly tools

- Setup internal structures
- Create rendezvous file(s) for tool support
- Note: servers have access to all client, tool functions
Rendezvous File Locations

System TMPDIR
- pmix.sys.host

Server TMPDIR (per nspace)
- pmix.host.tool
- pmix.host.tool.nspace
- pmix.host.tool.pid

Node B
- System PMIx server
- Mpirun
- rndvsFile

PRRTE demo
PMIx server initialization options:

- Process ID, system and server tmpdir
- Accept tool connections?
- Act as “system server” on that node?
- Server backend function module
  - Can be NULL or empty

Code snippet:

```
PMIx_server_init(pmix_server_module_t *module,
                 pmix_info_t info[], size_t ninfo)
```
Server Function Pointer Module

- Struct of function pointers (currently 26)
  - Provide access to host environment operations, info
  - Request support for inter-node ops
  - NULL or omitted => no support for that function

- Return rules
  - PMIX_SUCCESS: request accepted, cbfunc executed when complete
    - Cbfunc cannot be called prior to return from function
  - PMIX_OPERATION_SUCCEEDED: operation completed and successful, cbfunc will not be called
  - PMIx error code: problem with request, cbfunc will not be called
Module Functions

• **Client_connected**
  - Client has connected to server, passing all internal security screenings
    - Matches expected uid/gid, psec plugin checks
  - Server response: indicate if connection is okay, host support ready

• **Client_finalized**
  - Client has called PMIx_Finalize
  - Server response: allow client to leave PMIx
• Abort

- Client requests that specified procs be terminated and provided status/msg be reported to user
  - NULL proc array => all members of requestor’s nspace
  - Request does not automatically include requestor

• Fence_nb

- Execute inter-node barrier collecting any provided data
- Array of participating procs indicates which nodes will participate
  - Host required to translate proc to node location
  - Forms op signature: multiple simultaneous ops allowed, only one per sig
- Return all collected data to each participating server

**Module Functions**

```c
const pmix_proc_t *proc, void *server_object, int status, const char msg[], pmix_proc_t procs[], size_t nprocs, pmix_op_cbfunc_t cbfunc, void *cbdata
```

```c
const pmix_proc_t procs[], size_t nprocs, const pmix_info_t info[], size_t ninfo, char *data, size_t ndata, pmix_modex_cbfunc_t cbfunc, void *cbdata
```
Module Functions

• **Direct_modex**
  - Provide job-level data for nspace if rank=wildcard
  - Request any info “put” by the specified proc
  - Host required to:
    - Identify node where proc located
    - Pass request to PMIx server on that node
    - Return data response back to requesting PMIx server
Module Functions

- **Publish**
  - Publish information from source
  - Info array contains info + directives (range, persistence, etc.)
  - Duplicate keys in same range = error

- **Lookup**
  - Retrieve info published by publisher for provided keys (NULL -> all)
  - Info array contains directives (range)

- **Unpublish**
  - Delete data published by source for provided keys (NULL -> all)
  - Info array contains directives (range)
• Connect
  ▪ Record specified procs as “connected”
    • Treat failure of any proc as reportable event
  ▪ Collective operation
    • Array of procs => operation signature
    • Multiple simultaneous ops allowed, only one per signature

• Disconnect
  ▪ Separate specified procs
  ▪ Collective operation
    • Array of procs => operation signature
    • Multiple simultaneous ops allowed, only one per signature

Module Functions

const pmix_proc_t procs[], size_t nprocs, const pmix_info_t info[], size_t ninfo, pmix_op_cbfunc_t cbfunc, void *cbdata
Module Functions

- **Register_events**
  - Request host provide notification of specified event codes using PMIx_Notify_event API
    - NULL => all

- **Deregister_events**
  - Stop notifications for specified events
    - NULL => all

- **Notify event**
  - Request host notify all procs (within specified range) of given event code using PMIx_Notify_event

```c
pmix_status_t *codes, size_t ncodes, const pmix_info_t info[], size_t ninfo,
  pmix_op_cbfunc_t cbfunc, void *cbdata
```
Module Functions

- **Spawn**
  - Launch one or more applications on behalf of specified proc
  - Job-level directives apply to all apps, info provided to all procs
  - App-specific directives included in app object, info provided solely to app’s procs
  - Can include allocation directives

- **Listener**
  - Host shall monitor provided socket for connection requests, harvest/validate them, and call cbfunc for PMIx server to init client setup
• **Query**
  - Request information from the host environment (e.g., queue status, active namespaces, proc table, time remaining in allocation)

• **Tool_connected**
  - Tool has requested connection to server
    - Info contains uid/gid of tool plus optional service requests
    - Host can validate request, return proc ID for tool

```c
pmix_proc_t *proct, pmix_query_t *queries, size_t nqueries,
pmix_info_cbfunc_t cbfunc, void *cbdata

pmix_info_t *info, size_t ninfo,
pmix_tool_connection_cbfunc_t cbfunc, void *cbdata
```
Module Functions

- **Log**
  - Push the specified data to a persistent datastore or channel per directives
    - Syslog, email, text, system job log

- **Allocate**
  - Request modification to existing allocation
    - Extension (both time and resource), resource release, resource “lend”/”callback”
  - Request new allocation

```c
const pmix_proc_t *client, const pmix_info_t data[], size_t ndata,
const pmix_info_t directives[], size_t ndirs, pmix_op_cbfunc_t cbfunc, void *cbdata
```

```c
const pmix_proc_t *client, pmix_alloc_directive_t directive,
const pmix_info_t data[], size_t ndata,
pmix_info_cbfunc_t cbfunc, void *cbdata
```
Module Functions

- **Job_control**
  - Signal specified procs (pause, resume, kill, terminate, etc.)
  - Register files/directories for cleanup upon termination
  - Provision specified nodes with given image
  - Direct checkpoint of specified procs

- **Monitor**
  - Monitor this process for "signs of life"
    - File (size, access, modify), heartbeat, etc.
  - Failures reported as PMIx events

```c
const pmix_proc_t *requestor, const pmix_proc_t targets[], size_t ntargets,
const pmix_info_t directives[], size_t ndirs, pmix_info_cbfunc_t cbfunc, void *cbdata

const pmix_proc_t *requestor, const pmix_info_t *monitor, pmix_status_t error,
const pmix_info_t directives[], size_t ndirs, pmix_info_cbfunc_t cbfunc, void *cbdata
```
Module Functions

- **Get_credential**
  - Request a credential

- **Validate_credential**
  - Validate a credential

- **Group**
  - Perform a barrier op across specified procs
  - Perform any host tracking/cleanup operations
  - Return result of any special requests in directives
    - Assign unique context ID to group
Module Functions

- **IOF_pull**
  - Request the specified IO channels be forwarded from the given array of procs to this server for local distribution
  - Stdin is *not* supported in this call

- **Push_stdin**
  - Request the host transmit and deliver the provided data to stdin of the specified targets
    - Wildcard rank => all procs in that nspace
    - Source identifies the process whose stdin is being forwarded
Exercise 1: Create a Server

• Python or C – your choice
• Initialize a server
  ▪ Start with an empty server module
  ▪ Specify a “safe” tmpdir location
  ▪ Indicate it should be a “system” server
• Have it hang around
• Use “pattrs” to find out what it supports
• Add job_control function to server module
  ▪ Have it cause your server to exit
  ▪ Use PRRTE’s ”prun --terminate” to trigger it
Day 1: Detail

- Overview
- PMIx Reference Implementation
- Server Initialization
  - Exercise
- Launch Sequence
  - Exercise
Stage 0: Inventory Collection

- **Objective**
  - Gather a complete picture of all relevant hardware in the system
    - Utilizes HWLOC to obtain information
  - Allow each plugin to extract what is relevant to it
    - Fabric – NICs/HFIs plus distance matrix; topology, connectivity, and per-plane communication costs
    - Memory – available memory and hierarchy

- **Two collection modes**
• **PMIx_server_collect_inventory**
  - Collect inventory of local resources
  - Pass opaque blob back to host for transmission to WLM-based server
  - Info keys can specify types/level of detail of inventory to collect

• **PMIx_server_deliver_inventory**
  - Pass inventory blobs into PMIx server library for processing
  - Construct internal resource trackers

**Relevant Functions**

```c
pmix_info_t directives[], size_t ndirs,
pmix_info_cbfunc_t cbfunc, void *cbdata
```

```c
pmix_info_t info[], size_t ninfo,
pmix_info_t directives[], size_t ndirs,
pmix_op_cbfunc_t cbfunc, void *cbdata
```
Mode 1: Rollup

- RM Daemon
- PMIx_server_collect_inventory (default to local only)
- Inventory blob
- HWLOC
- Probe local inventory
- Filter thru plugins
- Extract NIC, memory info, etc
Mode 1: Rollup

WLM

PMIx_server_collect_inventory
(local+infra)

Obtain switch, connectivity, topology info

Construct internal resource trackers (plugins)

RM Daemon

PMIx_server_deliver_inventory

Inventory blob

Probe local inventory

Filter thru plugins

Extract NIC, memory info, etc

HWLOC

Filter thru plugins

Construct internal resource trackers (plugins)

“phone home”
Mode 2: Central

- PMIx_server_collect_inventory (global)
- Obtain NIC, switch, connectivity, topology info
- Construct internal resource trackers (plugins)

WLM

Only collects inventory accessible via centralized source (e.g., FM)

Option: WLM can request remote daemons respond with their local inventory
Stage 1: Scheduling

• Storage timing
  ▪ Identify dependencies
  ▪ Estimate caching/retrieval times

• Fabric considerations
  ▪ Access relative communication costs
    • Asynchronously updated by FM events
  ▪ Capabilities of each plane
    • Map user requests vs available planes
Baseline Storage Vision

• Tiered storage
  ▪ Parallel file system
  ▪ Caches at IO server, switches, cabinets, …
  ▪ Caches hold images, files, executables, libraries, checkpoints

• Bits flow in all directions
  ▪ Stage locations prior to launch
  ▪ Movement in response to faults, dynamic workflow, computational stages
Estimate Retrieval Times

User-specified caching, dependencies (data & libs), persistence

Parse for dependencies (plugins)

Query

Current data map
Usage patterns
Authorization

Retrieval time
Relevant Storage Functions

(signatures TBD)

- **Dependencies**
  - Support multiple methods via plugins
  - Typical ldd-like checks, others are active area of research

- **Accessibility**
  - List of files and uid/gid or credential, return accessibility status for each file
  - Include temperature/location (e.g., hot/cached, warm/on disk), other metadata

- **Scheduling data**
  - Time/cost to move specified files to given target locations (nodes, caches, temp)

- **Info queries**
  - Available storage, unit of reservation (block size)
  - Storage strategies (striping patterns)
  - Capabilities (QoS levels, bandwidth, topology, co-located processes)
• PMIx_server_register_fabric
  ▪ Obtain a handle to a specific fabric plane
  ▪ Can specify plane by characteristics or name
    • Obtain available names via PMIx_Query
• Pmix_server_deregister_fabric
  ▪ Release the fabric handle
• Terminology
  ▪ Vertex: NIC or switch interface, can include metadata
  ▪ Index: column or row in the cost matrix

Correspondence changes as interfaces fail, go offline, return as entire cost matrix is updated by FM!
Dealing With Updates

- Fabric plane handle tracks revision
- Matrix updates
  - Occur in thread-safe event
  - Increment matrix revision
- Functions that access cost data
  - Execute in thread-safe event
  - Check handle version against matrix version
  - Return PMIX_FABRIC_UPDATED if mismatch
- PMIx_server_update_fabric (pmix_fabric_t *fabric)
  - Syncs version level of handle to matrix
Relevant Fabric Functions

- **PMIx_server_get_num_vertices**  
  - pmix_fabric_t *fabric, uint32_t *nverts  
  - Get number of vertices in the provided fabric plane

- **PMIx_server_get_comm_cost**  
  - pmix_fabric_t *fabric, uint32_t src, uint32_t dest, uint16_t *cost  
  - Obtain relative communication cost for sending message from src to dest across provided plane

- **PMIx_server_get_vertex_info**  
  - pmix_fabric_t *fabric, uint32_t i, pmix_value_t *vertex, char **nodename  
  - Given index, get interface metadata and name of node/switch hosting it

- **PMIx_server_get_index**  
  - pmix_fabric_t *fabric, pmix_value_t *vertex, uint32_t *i, char **nodename  
  - Given vertex, get matrix index and name of node/switch hosting it

Open issue: query/return blocks of results – e.g., “give me 100 nodes with minimum relative comm cost”? May prove too complex a query due to number of constraint options.
Exercise 2: Scheduler Support

- Extend your previous server using the "test" fabric component
  - PMIX_MCA_pnet=test
  - PMIX_MCA_pnet_test_nverts=nodes:5;plane:d:3
- Collect the inventory
- How many NICs are in the system?
- Print the communication costs between them
- What vertex info is available for index 3?
- What is the index of the 1st NIC on node “test001”?
Stage 2: Launch Prep

• Storage requests
  § Request pre-position/cache data
  § Allocate storage resources

• Fabric requests
  § Obtain fabric info for application
    • Endpoints, network coordinates, etc.
  § Set fabric configuration
    • Software-defined topologies, QoS, etc.
  § Obtain security credentials

• Collect envars to forward
Storage Directives

- **Shift data**
  - Move cache to parallel file system to clear room
  - Pre-position data from file system to cache
    - Gateway, network-near target nodes, on-node bulk memory
  - Async operation – callback upon completion

- **Allocate storage resources**
  - Manage cache allotments
  - Set storage strategy for job

*(signatures TBD)*
Data Mover

Gateway Node

System PMIx server

User DM

Lustre

Cache

WLM

Data movement directives

Job Script

fork/exec
Setup Application

- **PMIx_server_setup_application**
  - Process mapping: What procs are on which nodes and where they are bound
  - Any directives regarding fabric settings (e.g., planes to be used, QoS), others

- **Cycle across active components**
  - Fabric plugins
    - Assign endpoints: info directives indicate how many per plane to assign to each proc, assignments provided in order of closest NIC to proc
    - Generate fabric credential(s) for job
    - Collect fabric-specific envars and settings for client libraries/drivers
  - Storage plugins
    - Alert job starting, retrieve storage settings for client libraries/drivers

- **Pickup PMIx-specific envars**
- **Return info array for delivery to compute nodes**
What About mpiexec?

- Launch its daemons on all nodes
  - Collect inventory from each
  - Proceed as before
- If inventory not available
  - PMIx_server_setup_application automatically requests info from scheduler
    - Provide URI for scheduler PMIx server
    - “Upcall” to RM for transmission
Exercise 3: Launch Prep

- Extend your previous server
  - PMIX_MCA_pnet=test
  - PMIX_MCA_pnet_test_nverts=nodes:5;plane:d:3
- Define an application (keep it simple)
  - Hosts: “test000,test001,test002”
  - Ppn: “0,1,2;3,4,5;6,7”
  - Remember to use the regex generators!
- Setup the application
  - Allocate network resources and security key
  - Pickup all related envars
  - Use the PNET verbosity parameter to see what it is doing
- Print out the result
Stage 3: Local Spawn Prep

- Extract setup array from launch msg
  - Check for job-level directives
    - Modify paths, set/unset env vars
  - PMIx_server_setup_local_support
    - Pass input to all active components
    - Fabric plugins
      - Setup local drivers, prep address tables, …
    - Storage plugins
      - Setup local drivers, configure memory, …
  - PMIx_server_register_nspace
    - Pass in job- and proc-level info for clients
    - Include setup array info, process map

```c
const pmix_nspace_t nspace,
const pmix_nspace_t nspace, int nlocalprocs,
const pmix_info_t info[], pmix_info_t info[], size_t ninfo,
const pmix_info_t info[], size_t ninfo,
const pmix_op_cbfunc_t cbfunc, void *cbdata
const pmix_op_cbfunc_t cbfunc, void *cbdata
```
Stage 4: Fork/Exec

- **PMIx_server_register_client**
  - Register each *local* proc for this nspace
  - Informs server of expected uid/gid of connecting client for security check
  - Server preps client support infrastructure

- **PMIx_server_setup_fork**
  - Add PMIx server connection and support info to env
  - Add subsystem-specific envvars for client libraries (e.g., fabric, storage)
Stage 5: Process Startup

- Handshake with server
  - Sets compatibility plugins
  - Server function module
    - Given chance to validate or reject connecting client
- Transfer data to client
  - Setup SM datastore
  - Send copy to client
Exercise 4: Fork/Exec Prep

- Extend your previous server
- Setup the local support
  - Pass in the data returned by setup application
  - Use the GDS and PNET verbosity parameters to see what it is doing
- Register the nspace
  - For now, just pass universe size and 3 local procs
- Register the local clients
- Setup the fork environment for each client
- Print out the results
• PMIx_server_deregister_client
  ▪ Called when local client terminates
    • Often called from within function module client_terminated entry
    • Both normal and abnormal termination
  ▪ Provides server library with chance to cleanup

• Generate event
  ▪ Abnormal termination only to avoid floods
    • Typically only upon request included with spawn directives
  ▪ Notify anyone listening for PMIX_PROC_ABORTED event
    • Provide ID of affected proc, any provided text message and/or info
    • Target only nspace of affected proc unless otherwise directed
    • Target non-default handlers
Stage 7: Job Termination

- **PMLx_server_deregister_nspace**
  - Called when job completes
    - Note: PMLx cannot provide function module entry as it doesn’t see multi-node job status
  - Provides server library with chance to cleanup

- **Generate event**
  - Notify anyone listening for PMIX_JOB_TERMINATED event
    - Optional to perform by default
  - Target non-default handlers
  - Provide exit status, any associated text message and/or info
Day 1: Detail

- Overview
- PMIx Reference Implementation
- Server Initialization
  - Exercise
- Launch Sequence
  - Exercise
Exercise: Scheduler

• Extend your server to support a scheduler
• Collect local inventory
• Poke around the comm cost matrix
  ▪ Perhaps with "pquery" tool?
• Define an application and set it up
  ▪ Set pnet_base_verbose=100 to see what it does
Agenda

• Day 1: Server & Scheduler
  ▪ Overview of PMIx
  ▪ Detailed look at Launch
• Day 2: Client, Tools, & Events – Oh My!
  ▪ Event notification
  ▪ PMIx Client functions
  ▪ PMIx Tool support
Events

- Async notification
  - Proc failures, system issues, coordination requests, workflow orchestration

- Types of events
  - Job-specific: directly relate to executing job
    - Debugger attachment, proc failure, app-generated event
    - Always delivered to the PMIx server by host
  - Environment: indirectly relate to a job but not specifically targeting it
    - ECC errors, temperature excursions, …
    - Delivered only upon request to host

- Event codes
  - Any integer value
  - Host-specific values must be either positive or lie beyond PMIX_EXTERNAL_ERR_BASE
Anyone can register
  - Host subsystem elements, apps, tools

**PMIx_Register_event_handler**
  - Specify any number of codes (3 categories)
    - NULL => default handler for all codes
    - Single code, Multiple codes
  - Can provide string name for this handler
    - Used for ordering and debugging
  - Callback returns handler registration ID (deregister, returned in notifications)
  - Handlers not required to be unique (can register same function multiple times)

Event caching
  - Job-specific events *required* to be cached and delivered in order
  - Environment events are *requested* to be cached

```c
pmix_status_t codes[], size_t ncodes,
pmix_info_t info[], size_t ninfo,
pmix_notification_fn_t evhdlr,
pmix_hdlr_reg_cbfunc_t cbfunc,
void *cbdata
```
**Handler Directives**

- **Specify ordering at time of registration**
  - First => execute this handler before any others*
  - Last => execute this handler after all others have completed*
  - First in category => execute this handler before any others for the event category*
  - Last in category => execute after all handlers for the event category have completed*
  - Before – insert immediately before the named handler
  - After – insert immediately after the named handler
  - Prepend – add to the front of the list for this category
  - Append – add to the end of the list for this category

- **Restrict interest**
  - Pass array of specific affected procs we want to hear about
  - Events impacting all other procs will be ignored for that handler

*only one of each
Handler Signature

ID of handler being called

Event code

Proc that generated event

size_t evhdlr_registration_id,
pmix_status_t status,
const pmix_proc_t *source,
pmix_info_t info[], size_t ninfo,
pmix_info_t *results, size_t nresults,
pmix_event_notification_cbfunc_fn_t cbfunc,
void *cbdata);

Info provided by source

Aggregate of results from all prior handlers

Handler return code

pmix_status_t status,
pmix_info_t *results, size_t nresults,
pmix_op_cbfunc_t cbfunc, void *thiscbdata,
void *notification_cbdata

Array of results from this handler

Callback fn/data to release handler data
• Anyone can generate an event
  ▪ Application procs, tools, host

• PMIx_Notify_event
  ▪ Report a single event code plus source that generated the event
  ▪ Specify a delivery range
    ▪ RM: solely to the host
    ▪ Local: available to procs on local node only
    ▪ Namespace: available to procs in same nspace only
    ▪ Session: available to procs in same session only
    ▪ Global: available to all procs
    ▪ Proc_local: available only internally to the generating proc
    ▪ Custom: array of specific target procs
  ▪ Provide additional info
    ▪ Affected proc(s), do not deliver to default event handlers

```c
pmix_status_t status,
const pmix_proc_t *source,
pmix_data_range_t range,
const pmix_info_t info[], size_t ninfo,
pmix_op_cbfunc_t cbfunc, void *cbdata
```
Event Handling

• Precedence order
  ▪ First
  ▪ Single code -> Multi-code -> Default handlers
    • First/last called in each category
  ▪ Last

• Results “chained”
  ▪ Results returned by each handler are added to end of results array passed to next handler

• Each handler must call event handler completion function
  ▪ All processing stops upon return of PMIX_EVENT_ACTION_COMPLETE
  ▪ Not allowed to perform any blocking operation during handler
Event Notes

- Last handler is called after all registered default handlers matching specified range
  - Ensure no default handler aborts process before it
- Events cannot be delivered back to the process that generated them
  - Host cannot pass event back to its PMIx server library
  - Server library cannot pass event back to generating client
- Keep event handlers short
  - PMIx server library is “blocked” until completion
Send and then internally process
Example Uses

• Hybrid applications
  ▪ Notify programming libraries of each others existence, operations
  ▪ OpenMP + MPI: coordinate programming blocks
  ▪ Notification strictly within the individual proc

• Fault tolerance: ULFM
  ▪ Notification of process failure

• Tools
  ▪ Notification of job completion
  ▪ Debugger attachment handshake
Query App Info

PMIx Server

mpiexec

pmid

Proc

Dbgr Dmn

Register handler for event

App query event

Query

Query
• Day 1: Server & Scheduler
  ▪ Overview of PMIx
  ▪ Detailed look at Launch
• Day 2: Client, Tools, & Events – Oh My!
  ▪ Event notification
  ▪ PMIx Client functions
  ▪ PMIx Tool support
PMIx Scope: Client

- **Wireup**
  - Fence, put, get, commit
- **Publication**
  - Publish, lookup, unpublish
- **Dynamics**
  - Spawn, connect, disconnect, group construct/destruct
- **Storage**
  - Estimate retrieval times, set hot/warm/cold policy, data movement
- **WLM**
  - Inventory, comm costs, subsystem app resource allocations, allocation mgmt
- **Fabric**
  - QoS control, async updates
- **Tools**
  - Query, attach/detach, IO fwd
- **Events (async notification)**
- **Info**
  - Query, logging
• PMIx_Put
  ▪ Adds provided key-value pair to internal cache
  ▪ Duplicate keys are overwritten
• PMIx_Commit
  ▪ Sends all added/modified key-value pairs since last commit to local PMIx server
  ▪ Server required to store keys on per-proc basis – i.e., procs can post the identical key without overwrite

• Fence
  ▪ Barrier operation
  ▪ Data collection optional
    ▫ const pmix_proc_t procs[], size_t nprocs,
    ▫ const pmix_info_t info[], size_t ninfo
• Get
  ▪ Retrieve key for a given proc
    ▫ PMIX_RANK_UNDEF: retrieve globally unique key (legacy support)
    ▫ Check internal/shmem first
    ▫ Request from server
      ▫ Obtain from remote server hosting specified proc if data not exchanged
Key-Value “Scope”

Who can “Get” this key-value pair?

- Specified by source process at time of “put”
- Controls access by other procs
  - “internal”: only available to the source proc
  - ”local”: only accessible by other procs on same node
  - “remote”: only available to procs on other nodes
  - “global”: available to everyone
- Only remote and global scope included in data exchanges during “fence”
Publication

- **PMLx_Publish**  
  `const pmix_info_t info[], size_t ninfo`
  - Publish data in info array to specified range (default: session)
  - Keys must be unique within given range
    - Not indexed by source proc!
    - First published “wins” – followers return error
  - Persistence instructs server as to how long data is retained (default: app)
- **PMLx_Unpublish**  
  `char **keys, const pmix_info_t info[], size_t ninfo`
  - Delete data for specified keys
  - NULL => delete all data published by this process

- **PMLx_Lookup**  
  `pmix_pdata_t data[], size_t ndata, const pmix_info_t info[], size_t ninfo`
  - Retrieve published data
  - Constrained to data published by current uid/gid
  - Returns error if not found
    - Optional: wait for first found data, wait for all data, timeout
  - “non-found” data will have PMIX_UNDEF datatype
Range & Persistence

• **Range:** who has access to data
  - “proc_local”: only within the proc itself (e.g., across threads)
  - "local": only procs on local node
  - “namespace”: only procs within same nspace (job) as publisher
  - “session”: only procs within same session (allocation) as publisher
  - “global”: any process
  - “custom”: only specified processes
  - “rm”: only the host environment

• **Persistence:** when data shall automatically be deleted
  - “first_read”: delete after first access
  - “proc”: retain until publisher terminates
  - “app”: delete when publisher’s application terminates
  - “namespace”: delete when publisher’s nspace (job) terminates
  - “session”: delete when publisher’s session (allocation) terminates
  - “indef”: retain until specifically deleted
Dynamics: Basic

- **Spawn**
  - Spawn new job
    - Job_info specifies directives and info for all apps
    - Apps array contains info for each individual app
  - Namespace returned upon spawn complete
  - Variety of notification options
    - Job launched, job terminated, app terminated, proc terminated

- **Connect**
  - Mark the specified procs as “connected”
  - All procs to receive
    - Job-level info for nspaces of all participants
    - “put” info from participants, filtered by scope

- **Disconnect**
  - Remove “connected” specification for given procs
  - Return error if not connected
• Relation to RM
  ▪ Connect: passed to RM, no new ID assigned
  ▪ Group: handled by PMIx server, each proc assigned new “group rank”, translate group IDs to global IDs for RM operations

• Construction
  ▪ Connect: bulk synchronous only
  ▪ Group: can be dynamic, invite/join as well as nonblocking

• Destruction
  ▪ Disconnect: bulk synchronous only
  ▪ Group: can be dynamic, members notified as procs leave
• PMIx_Allocation_request
  ▪ Request allocation of new resources
  ▪ Extend current reservation on specified resources
  ▪ Release current specified resources
  ▪ “Lend” resources back, mark for return on request or after specified time
    • Return requested by passing PMIX_ALLOC_REAQUIRE directive

• RM can notify of resource changes
  ▪ Registration for event required
Job Control & Monitoring

- **PMIx_Job_control**
  - Include string ID with request
    - Allows later query for status, cancellation of request
  - Signal, kill, politely terminate
  - Direct targets to checkpoint
    - PMIx event, signal, etc
  - Provision specified nodes with indicated image
  - Register files and directories for cleanup after termination
  - Register willingness to be preempted

- **PMIx_Process_monitor**
  - Monitor file changes (access, mod, size)
  - Heartbeat
Information

- **PMIx.Resolve_nodes**
  - Given nspace, return comma-delimited list of nodes hosting procs within it

- **PMIx.Resolve_peers**
  - Given node, return array of procs within given nspace on it (NULL => all)

- **Query**
  - Request supported APIs, attributes
  - Executing jobs, process tables, queue status
  - Psets, groups, available resources

- **Log**
  - Deliver provided message to one or more logging channels
  - Syslog (local, global), email, text, global data store, job record
Security & Storage

• Get/validate credential
  ▪ Some built-in support for credential services
    • Munge, Cray DRC
  ▪ Others passed to host for servicing

• Storage
  ▪ Data movement, storage strategies, availability and location
• Day 1: Server & Scheduler
  ▪ Overview of PMIx
  ▪ Detailed look at Launch
• Day 2: Client, Tools, & Events – Oh My!
  ▪ Event notification
  ▪ PMIx Client functions
  ▪ PMIx Tool support
Tool Support Examples

- **Query**
  - Network topology
    - Array of proc network-relative locations
    - Overall topology (e.g., “dragonfly”)
  - Running jobs
    - Currently executing job namespaces
    - Array of proc location, status, PID
  - Resources
    - Available system resources
    - Array of proc location, resource utilization (ala “top”)
  - Queue status
    - Current scheduler queue backlog

- **Event injection**
  - Async directives to running jobs

- **Storage directives**
  - Move/delete files between storage locations

- **Job submission**

- **Debuggers**
  - Portable attach, query mechanism
Tool Basics

- **Two types**
  - Client
    - Launched by a PMIx server – has identifier
  - Launcher
    - Will be spawning processes – e.g., “mpiexec”
    - May or may not also be client

- **Servers must “opt in” for tool connection support**
  - PMIX_SERVER_TOOL_SUPPORT – allow support
  - PMIX_SERVER_REMOTE_CONNECTIONS – allow remote connections
  - PMIX_SERVER_SYSTEM_SUPPORT - system server (max one/node)
  - Job-specific server (default)
Tool Connections

Only one connection at a time!
Rendezvous File Locations

System TMPDIR
- pmix.sys.host

Server TMPDIR (per nspace)
- pmix.host.tool.nspace
- pmix.host.tool.pid
- pmix.host.tool

Node B
- System PMIx server
- RM
- Mpirun
- rndvsFile

PRRTE demo
Tool Initialization

- **PMLx_tool_init**
  - Type of tool
  - Connection options
    - Do not connect
    - Connect via precedence rules

- **PMLx_tool_connect_to_server**
  - If connected, disconnect from current server
  - Connect to new server per precedence rules
Connection Precedence

- Given specific URI or filename
  - Special names found in configuration file (MCA param)
    - PMIX_CONNECT_TO_SCHED
- System server
  - If system-server-only, then return error if not found
- Scan server tmpdir’s
  - Given server PID or nspace
    - Returns error if not found or not allowed to access
  - First generic tool uid/gid allowed to access
Tool Connections: Remote

- Query local server for URI
  - Reconnect to returned URI
  - System and job-level servers
- Compute from configuration, given target node
  - MCA param for static socket of system servers
- Spawn proxy to scan
  - Assumes permissions and mechanism for spawn
General Capabilities

• Query RM or launcher for support
  ▪ Mechanisms for “hold” and “release”
  ▪ Daemon co-launch capabilities
  ▪ IO forwarding support

• Specify app release mechanism
  ▪ PMIx event, signal, …

• Register for events
  ▪ Termination of debugger job and/or daemons
  ▪ Termination of app job and/or procs
  ▪ Request debugger start on event from app
Debugger/Tool Features

• Co-launch/co-location of daemons
  - At initial app spawn
    • Co-launch
  - Upon attach
    • Spawn w/co-location

• Launch control
  - Stop-on-exec, stop-in-init, stop-in-app
  - Release method to be used

• Forwarding of IO
  - To/from debugger daemons
  - To/from app being debugged

• Query app info
  - Global and local proctable
  - Application internal metadata

• Direct/indirect launch support
  - Forward, set/unset/modify envars (e.g., LD_PRELOAD)
  - Launcher directives
    • Modify local fork/exec agent
    • Replace launcher daemons

• Local launcher fork/exec option
  - If PMIx_Spawn not available or if desired
Direct Launch

- Co-launch
- Two-stage launch
Indirect Launch

RM

PMIx_Server

PMIx_Spawn

mpiexec

PMIX_LAUNCHER_PAUSE_FOR_TOOL

fork/exec
Indirect Launch

Pass directives, application description
Indirect Launch

- **Co-launch**
- **Two-stage launch**
Attach to Running Job
Attach to Running Job

Direct or Indirect Launch
Assigning Procs->Daemons

Query:
- Local proctable
- Local rank

PMIX_DEBUG_JOB
Assigned in launch data
Assigning Procs->Daemons

Query:
- Local proctable
- Local rank

PMIX_DEBUG_JOB
Assigned in launch data
Assigning Procs->Daemons

Query:
- Local proctable
- Local rank

PMIX_DEBUG_JOB
Assigned in launch data
Forwarding Stdin

RM
PMIx
Server

RM
PMIx
Server

Proc
stdin
Sent via PMIx

PIMx Client Collects

Tool Collects

Dbgr
Dmn

stdin
Wrap-Up

• Covered a lot of ground
  ▪ Primary focus on scheduler

• Implementation status
  ▪ Client & basic server: in production
  ▪ Scheduler & fabric: alpha
  ▪ Storage: in definition

• Expected completion
  ▪ Release v4.0 in 2Q2020

Thank You!