A Scalable PMIx Database

Artem Y. Polyakov
Mellanox Technologies
artemp@mellanox.com

Joshua Ladd
Mellanox Technologies
jladd@mellanox.com

Elena Shipunova
Intel, Inc.
elena.shipunova@intel.com

Boris I. Karasev
Mellanox Technologies
boriska@mellanox.com

ABSTRACT

This work presents a scalability analysis of a PMIx Database. It is demonstrated that the main limiting factor is the scalability of the locking subsystem. A new scheme called *2N-lock* is proposed that demonstrates two orders of magnitude improvement in the PMIx *Get* latency.

CCS CONCEPTS

• **Computing methodologies** → **Concurrent algorithms**; • **Software and its engineering** → **Distributed systems organizing principles**; **Parallel programming languages**; *Message oriented middleware*;

KEYWORDS

Process Management Interface, HPC middleware, MPI, Resource management

ACM Reference Format:
Artem Y. Polyakov, Joshua Ladd, Elena Shipunova, and Boris I. Karasev. 2018. A Scalable PMIx Database. In *Proceedings of EuroMPI 2018 Conference (EuroMPI'18).* ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

A Process Management Interface (PMI) is the component of the HPC software stack that is responsible for interaction between a Resource Manager (RM) and a parallel application. In all PMI versions existing to date, the informational exchange between RM and application is organized in the form of a key-value database (KVDb) that has *Put*, *Get* operations and API-specific synchronization primitives.

The PMI Exascale (PMIx) (pseudo)standard [2] provides advanced capabilities to enable efficient bootstrapping of applications on emerging exascale systems. The most important distinguishing features of PMIx compared to its predecessors, PMI1 and PMI2, are: a) extended job-level information that allows exposing RM knowledge about a job environment that is typically required during an application startup, and b) *on-demand* exchange of the application-specific data (commonly referred to as *direct modex* in PMIx community).

This paper focuses on the problem of scalable distribution of the job-level and application-specific data from PMIx server (that represents RM) to PMIx client at the application side.

EuroMPI'18, September 2018, Barcelona, Spain

2 RELATED WORK

Intra-node KVDb access is known to be one of the major limitations of existing PMI implementations. It mainly affects the performance of a *Get* primitive. Chakraborty et al. [3] analyzed the scalability of the PMI2 implementation in Slurm RM and identified the message-based client-server communication as a significant bottleneck. They further explore the benefits of a shared memory mechanism to implement KVDb and conclude on its key advantages:

- *Scalable memory consumption*: KVDb is stored once per compute node as opposed to traditional replication of it on each application process.
- *More parallelism* as shared memory allows clients to access KVDb independently without server involvement.
- *Low latency* as operations can transact at the speed of CPU without requiring expensive system calls.

3 PMIX GET OPERATION DETAILS

PMIx relaxes synchronization assumptions that are guaranteed in PMI1 and PMI2. In particular, it allows on-demand information fetching which requires an extension of the approach described in [3] with lock-based KVDb access coordination.

The *PMIx_Get* fast-path algorithm is implemented as follows:

(1) *Perform a thread shift*; this step assumes the transfer of control to the PMIx service thread to ensure the thread safety (only service thread manages a global PMIx state and performs communications).
(2) *Lock* KVDb for reading.
(3) Attempt to *fetch* the requested key from the shared memory.
(4) *Unlock* KVDb.

For PMIx *Get* latency evaluation we used a 20-core IBM POWER8 system (two 10-core processors with 8 hardware threads per core, 160 threads total), with logical CPUs represented by hardware threads.

PMIx version 2.1 was used as the code base. In order to estimate PMIx primitives performance, we developed a

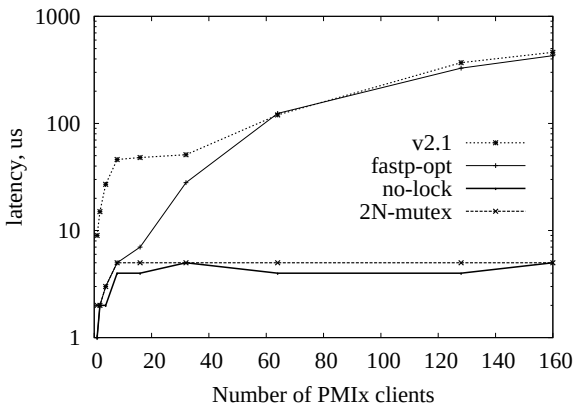

Figure 1: PMIx_Get latency on POWER8 system

microbenchmark called *pmix_perf* that is included in PMIx distribution starting from version 2.0. Processes were mapped by adjacent logical CPUs yielding gradual filling of a certain hardware resource before using to the next one.

The curve *v2.1* on fig. 1 represents the growth of *PMIx_Get* operation latency on POWER8 system where a level of 460 us is reached on a fully-occupied node.

4 CODE PATH OPTIMIZATIONS

Profiling the *PMIx_Get* implementation shows that most of the time is spent on locking steps (2 and 4). However, we start with a set of code cleanup optimizations of obvious inefficiencies in step (3) in order to isolate and attribute subsequent improvements to an improved locking scheme.

First, we removed thread shifting (1) from the fast path of the *Get* algorithm as the shared memory component does not access the global state of PMIx. *Second*, we eliminated extra memory allocations on the critical path replacing them with pre-allocated objects provided by *PMIx_Get* caller.

The curve *fastp-opt* (fig. 1) corresponds to PMIx v2.1 updated with these optimizations. For up to 8 clients (one core boundary) the performance is improved significantly and comparison with the lockless version (curve *no-lock*, fig. 1) confirms that one-core performance is very close to an optimal. However, as an application scales out of a core the advantage rapidly decreases and performance tends to the level of *v2.1*.

5 LOCKING OPTIMIZATION

PMIx utilizes Read/Write locks (RW-locks) to ensure that a client's read accesses are consistent with the server-side updates (write access). As was demonstrated in the previous section, pthread RW-lock implementation currently used in PMIx is not scalable.

The problem of scalable RW-locks is well known [1, 4]. However, PMIx database has several characteristics that distinguish it from the general problem.

- KVDb has only one writer (PMIx server) thus no arbitration between multiple writers is required.
- Write locks are only present in PMIx *on-demand* mode where only a few keys expected to be exchanged.
- Readers are typically assigned on execution units (cores or hardware threads) while the writer does not have dedicated hardware resource.
- Readers requesting the data are blocked waiting for the completion on the out-of-band channel.

Based on these observations, we prioritize a reader scalability attribute and propose the *2N-lock* scheme derived from the *static* approach described in [4]. The key difference of the *2N-lock* scheme is that it implements a writer-preference policy typical for PMIx scenario. Each reader has an individual pair of locks (μ_i and ν_i). ν_i is used by a writer to prevent new readers from getting the lock. The *2N-lock* has the following acquisition logic assuming that the number of readers is N:

Writer: 1) $lock(\nu_i), i \in [1, N]$; 2) $lock(\mu_i), i \in [1, N]$.

ith *reader*: 1) $lock(\mu_i)$; 2) $trylock(\nu_i)$; 3) if ν_i is acquired - $unlock(\nu_i)$, the lock is taken, else - $unlock(\mu_i)$ and block on ν_i giving the writer a precedence.

Compared to the pthread-based RW-locks, *2N-lock* avoids cache-coherency traffic in read-dominated workloads [1].

The curve *2N-lock*, fig. 1 shows that the latency of the new scheme is similar to a lockless configuration. Depending on readers contention level, the write lock latency varies in the $[0.05, 1.2]$ ms range. Considering the KVDb specifics outlined above, a low contention level is expected in a real scenario. We verified that using *pmix_perf* tool (*on-demand* mode) on a 32-node Intel x86_64 system with 28 cores per node.

In general, *2N-lock* provides a lockless-level latency for the full-data exchange mode while maintaining reasonable performance characteristics for the *on-demand* mode.

6 CONCLUSIONS

In this work, we analyzed the performance and scalability of PMIx Database. The main conclusions are:

- Pthread implementation of read/write locks has limited scalability.
- Second-level limiting factors: thread shifting and dynamic memory allocations on the *PMIx_Get* fast-path, were identified and eliminated.
- Finally, a new scalable locking scheme called *2N-lock* was proposed. It is planned for inclusion in PMIx 2.2.0.

REFERENCES

[1] I. Calciu, D. Dice, Y. Lev, V. Luchangco, V.J. Marathe, and N. Shavit. 2013. NUMA-aware reader-writer locks. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, New York, NY, USA, 157–166. https://doi.org/10.1145/2442516.2442532

[2] Ralph H. Castain, David Solt, Joshua Hursey, and Aurelien Bouteiller. 2017. PMIx: Process Management for Exascale Environments. In *Proceedings of the 24th European MPI Users' Group Meeting (EuroMPI '17)*. ACM, New York, NY, USA, Article 14, 10 pages. https://doi.org/10.1145/3127024.3127027

[3] Sourav Chakraborty, Hari Subramoni, Jonathan Perkins, and Dhabaleswar K. Panda. 2016. SHMEMPMI – Shared Memory Based PMI for Improved Performance and Scalability. In *Proceedings of the 24th European MPI Users' Group Meeting (CCGrid)*. IEEE, 60–69. https://doi.org/10.1109/CCGrid.2016.99

[4] W.C. Hsieh and W.E Weihl. 1992. Scalable Reader-Writer Locks-forParallel Systems. In *Proceedings of the Sixth International Parallel Processing Symposium*. IEEE, 656–659. https://doi.org/10.1109/IPPS.1992.222989