Extending MPI to Better Support Multi-Application Interaction

Jay Lofstead[1], Jai Dayal[2]

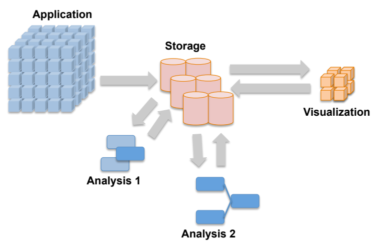
[1] Sandia National Laboratories
[2] Georgia Institute of Technology

Abstract. Current scientific workflows consist of generally several components either integrated *in situ* or as completely independent, asynchronous components using centralized storage as an interface. Neither of these approaches are likely to scale well into Exascale. Instead, separate applications and services will be launched using online communication to link these components of the scientific discovery process. Our experiences with coupling multiple, independent MPI applications, each with separate processing phases, exposes limitations preventing use of some of the optimized mechanisms within the MPI standard. In this regard, we have identified two shortcomings with current MPI implementations. First, MPI intercommunicators offer a mechanism to communicate across application boundaries, but do not address the impact this operating mode has on possible programming models for each separate application. Second, MPI_Probe offers a way to interleave both local messaging and remote messages, but has limitations as MPI_Bcast and other collective calls are not supported by MPI_Probe thus limiting use of optimize collective calls in this operating mode.

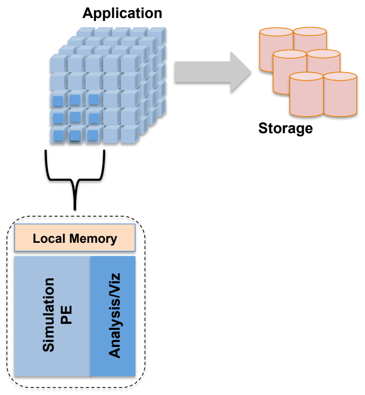
1 Introduction

The move toward exascale is changing how the scientific computing process works. Currently, one of two approaches is used. Most commonly, separate, independent applications are combined into a single process with scripting or workflow software to ease connecting the output from one component with another as illustrated in Figure 1(a). In a production environment, this is nearly exclusively done using a centralized storage system shared between pairs of connecting components. In general, this is a single storage system. In this approach, each component can scale independently, but is at the mercy of the file system performance for end-to-end scalability. Alternatively, applications can incorporate additional processing pieces, such as analysis or visualization components, in situ as illustrated in Figure 1(b). In this case, these additional processing pieces must scale as easily as the host simulation or scaling the combination will be artificially limited.

The alternative approach to address both of these cases is to use the best of each while avoiding the penalties of both. By using online data processing areas (see Figure 2) to create an online workflow, typically called data staging, hybrid



(a) Offline Processing



(b) In Situ Processing

Fig. 1. Traditional Scientific Workflow Architectures

staging, or 'in flight' processing, the speed penalty of a centralized file system is avoided and the scalability limitations of a single, integrated, executable are avoided. This approach, while not without its own challenges, has proven to work well for both tightly coupled and loosely coupled workflows.

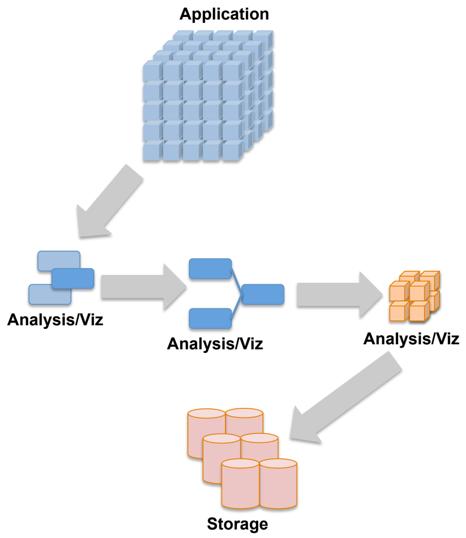


Fig. 2. Online Processing

MPI provides the concept of an *inter-communicator* to connect two applications with MPI messages. This feature works well enough for the inter-application communication, but MPI does not adequately address the potential impact of connecting applications on their operating model. Each application, particularly shared services-style applications, offers a collection of operations to process data. For shared services applications, one or more of these operations may be in process or at least a possible next step at any time. This requires the external interface to seamlessly support any possible message at any time including the corresponding memory use for each of these messages. For all of these cases, if any currently processing task issues a collective call, all processes must participate for the program to continue. The serious implication is that being responsive to messages from the other application requires periodic checks for waiting messages. For a services-style application, this external communication may drive what local processing is performed by different portions of the local application. To optimize the communication between applications, it becomes more efficient to probe for the inter-application messages as well as the local messages on the communicating processes. The lack of complete support of message types in the MPI_Probe call limits the kinds of operations that can be used within these applications.

The rest of the paper is structured as follows. In Section 2 we discuss some of the related work as well as work that motivates the need for these changes. Section 3 discusses the current design and the implications of these decisions for programming of participating application components. The proposed solution is presented next in Section 4. Finally, conclusions are presented in Section 5.

2 Related Work

Separate MPI applications operating as a single workflow using isolated applications to isolate failures has been demonstrated previously. The C-MPI project [11] uses DHTs to connect MPI applications. The LDM [10] offers a similar approach to data staging techniques as demonstrated in the LEAD [2] project. In this case, the various data processing components are linked together to form the processing workflow.

Offline workflows have been built using a variety of tools. For example, Dagman [5], Pegasus [7], and Kepler [4] each provide a way to connect various components in an ordered way to process scientific data. Scientists have also assembled similar systems less formally using scripts. They each work by providing a way to trigger a component at a given time given a set of conditions, such as a prior dependency component has completed processing. In all of these cases, the use of centralized storage as an integration point introduces a performance bottleneck.

The alternative approach of in situ processing, such as is done by ParaView [6] and VisIt [9], has its own problems. For example, the CTH [3] shock physics code in use at Sandia easily can scale to 100,000 cores with an executable size of around 30 MB. When incorporating ParaView for in situ processing and visualization, the executable grows to around 300 MB and has difficulty scaling beyond around 30,000 cores. While ParaView is actively working to correct these scaling limitations, those fixes will not solve the increased memory footprint fully.

PreDatA [12] offers 'in flight' data processing from the simulation to disk by hosting the processing in various locations along the data path. Alternatively, DataSpaces [1] and the related projects at Rutgers focus on attempting to store data in an online repository for querying by another application. These approaches currently rely on custom connections between components and do not offer the portability offered by MPI intercommunicators. The Network Scalable Services Infrastructure [8] offers an RPC-style interface to efficiently connect between separate applications using native interconnect techniques, but offers no default services.

3 Current Design Attributes

Current applications that wish to participate in an online workflow would incorporate some minor changes. For example, an additional interface that communicates with the related applications is added and then incorporated into the processing loop. This simplifies the application changes by limiting the number of places where code to check for inter-application messaging may occur. This isolates these changes while allowing the application to run as it is originally written. While this is simple in itself, scenarios such as when there is active processing in one or more of the participating applications causes problems. The difficulty with message probing is the lack of support to detect any collective operations. For our motivating example, a mass data transfer from one application to another requires some configuration information to be sent across the inter-application control message interface to indicate the number of variables, their types, extents, and data types. The amount of data in this message is unknown and only slightly bounded. Another message type coming across the same interface will be a variable itself being sent across the application boundary. In this case, the size could be as much as 10% or more of the node's local memory. The kinds of processing that may occur for a variable may require that all pieces of the global variable are processed simultaneously to generate some summary or derived value. Frequently these operations are performed using collective calls as part of a larger processing sequence.

Efficiency strongly suggests that the configuration information is sent across once and distributed out to participating processes locally. This distribution of messages among the processes of one application is generally performed using an MPI_Bcast or similar mechanism to take advantage of the optimizations incorporated into the MPI standard. The difficulty here is two fold. First, the processes operate out of step with each other, so not all subsets of processes will expect the same types of messages. Second, while under the proposed MPI 3.0 standard, it is possible for processes to pre-post for asynchronous collective operations, it is assumed that the processes know what to expect a head of time. In the case of a general processing situation, it is unpredictable what the specifics of the collective call would be rendering this ineffective for this situation. Instead, to implement this functionality, a manual asynchronous broadcast must be implemented. This problem is worse if the processing incorporates other collective operations, such as all-to-all, gather, scatter, or all-reduce operations. None of these pending operations can be detected through the use of an MPI_Probe call.

4 Proposed Extensions

One possible solution to this scenario is relatively straightforward. The current MPI_Probe implementation could be expanded to include all of the collective calls. While this maintains a simpler API, the additional possibilities for types of messages and the change in the behavior that may affect current applications makes this approach less than desirable. Instead, an identical pair of API calls, MPI_Cprobe and MPI_Icprobe that look the same as the MPI_Probe equivalents, would be sufficient. In this case, instead of identifying pending point to point messages, these calls would only detect all of the collective calls that the current probe implementation supports. With this extension, it would be possible to remove re-implentation of collective calls as point-to-point calls and associated operations such as performing the all-reduce operation. Additionally, this would afford potentially leveraging hardware features, such as the collectives network on the BlueGene platform.

The potential performance impact of blocking many or even all processes waiting for a collective call to complete is serious. It is certainly likely that collectives were not included in MPI_Probe for exactly this concern. However, the introduction of asynchronous collective communication largely alleviates this concern. The potential performance penalty of poorly written replacements for the collective calls should outweigh these concerns. Their direct impact of these calls on an MPI application's performance will generally be limited. With sufficient warnings about only using these calls as ways to detect collective calls will cause all processes to stall until the corresponding collective calls are issued.

5 Conclusions

The move to exascale is motivating moving offline workflows online and coupling the various components more tightly while maintaining separate applications to enhance resilience. This communication and processing intensive software architecture requires the ability to both probe for new messages as well as communicate among all of the processes within an application simultaneously. The ability to probe for unexpected messages of all types rather than simply point-to-point messages will enable MPI applications to more easily participate in this software architecture. The inclusion of collective calls such as MPI_Bcast both simplifies the implementation as well as offers the performance advantage of efficient collectives implementations offered by MPI and potentially the ability to leverage hardware features such as dedicated collectives networks.

6 Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] Docan, C., Parashar, M., Klasky, S.: DataSpaces: An interaction and coordination framework for coupled simulation workflows. HPDC '10: Proceedings of the 18th international symposium on High performance distributed computing (2010)

[2] Droegemeier, K., Chandrasekar, V., Clark, R., Gannon, D., Graves, S., Joseph, E., Ramamurthy, M., Wilhelmson, R., Brewster, K., Domenico, B., Leyton, T., Morris, V., Murray, D., Plale, B., Ramachandran, R., Reed, D., Rushing, J., Weber, D., Wilson, A., Xue, M., Yalda, S.: Linked environments for atmospheric discovery (lead): A cyberinfrastructure for mesoscale meteorology research and education. In: 20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology. Seattle, WA (01/2004 2004), http://www.cs.indiana.edu/dde/papers/droegemeierIIPS2004.pdf

[3] Jr., E.S.H., Bell, R.L., Elrick, M.G., Farnsworth, A.V., Kerley, G.I., McGlaun, J.M., Petney, S.V., Silling, S.A., Taylor, P.A., Yarrington, L.: CTH: A software family for multi-dimensional shock physics analysis. In: Brun, R., Dumitrescu, L. (eds.) Proceedings of the 19'th International Symposium on Shock Physics. vol. 1, pp. 377–382. Marseille, France (July 1993), http://sherpa.sandia.gov/9231home/pdfpapers/issw.pdf

[4] Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E.A., Tao, J., Zhao, Y.: Scientific workflow management and the kepler system: Research articles. Concurr. Comput. : Pract. Exper. 18(10), 1039–1065 (2006)

[5] Malewicz, G., Foster, I., Rosenberg, A., Wilde, M.: A tool for prioritizing DAGMan jobs and its evaluation. High Performance Distributed Computing, 2006 15th IEEE International Symposium on pp. 156–168 (0-0 2006)

[6] Moreland, K., Lepage, D., Koller, D., Humphreys, G.: Remote rendering for ultrascale data. Journal of Physics: Conference Series 125(1), 012096 (2008), http://stacks.iop.org/1742-6596/125/i=1/a=012096

[7] Mullender, S.J., Leslie, I.M., McAuley, D.: Operating-system support for distributed multimedia. In: Proceedings of the 1994 Summer USENIX Technical Conference. pp. 209–219 (1994)

[8] Oldfield, R.A., Widener, P., Maccabe, A.B., Ward, L., Kordenbrock, T.: Efficient data-movement for lightweight I/O. In: Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems. Barcelona, Spain (September 2006), http://doi.ieeecomputersociety.org/10.1109/CLUSTR.2006.311897

[9] Riedel, M., Eickermann, T., Habbinga, S., Frings, W., Gibbon, P., Mallmann, D., Wolf, F., Streit, A., Lippert, T., Schiffmann, W., Ernst, A., Spurzem, R., Nagel, W.: Computational steering and online visualization of scientific applications on large-scale hpc systems within e-science infrastructures. In: e-Science and Grid Computing, IEEE International Conference on. pp. 483 –490 (dec 2007)

[10] UCAR: Local data manager, http://www.unidata.ucar.edu/software/ldm

[11] Wozniak, J.M., Latham, R., Lang, S., Son, S.W., Ross, R.: C-mpi: A dht implementation for grid and hpc environments. In: EuroMPI (2009)

[12] Zheng, F., Abbasi, H., Docan, C., Lofstead, J., Klasky, S., Liu, Q., Parashar, M., Podhorszki, N., Schwan, K., Wolf, M.: PreDatA - preparatory data analytics on Peta-Scale machines. In: In Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium, April, Atlanta, Georgia (2010)