SANDIA REPORT

SAND2015-5029
Unlimited Release
Printed June 2015

Delta: Data Reduction for Integrated Application Workflows

Gregory Jean-Baptiste, Gerald F. Lofstead, Ron A. Oldfield

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2015-5029
Unlimited Release
Printed June 2015

Delta: Data Reduction for Integrated Application Workflows

Gregory Jean-Baptiste
Florida International University
gjean011@fiu.edu

Gerald F. Lofstead
Center for Computing Research
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1319
gflofst@sandia.gov

Ron A. Oldfield
Center for Computing Research
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1319
raoldfi@sandia.gov

Abstract

Integrated Application Workflows (IAWs) run multiple simulation workflow components concurrently on an HPC resource connecting these components using compute area resources and compensating for any performance or data processing rate mismatches. These IAWs require high frequency and high volume data transfers between compute nodes and staging area nodes during the lifetime of a large parallel computation. The available network bandwidth between the two areas may not be enough to efficiently support the data movement. As the processing power available to compute resources increases, the requirements for this data transfer will become more difficult to satisfy and perhaps will not be satisfiable at all

since network capabilities are not expanding at a comparable rate. Furthermore, energy consumption in HPC environments is expected to grow by an order of magnitude as exascale systems become a reality. The energy cost of moving large amounts of data frequently will contribute to this issue. It is necessary to reduce the volume of data without reducing the quality of data when it is being processed and analyzed. Delta resolves the issue by addressing the lifetime data transfer operations. Delta removes subsequent identical copies of already transmitted data during transfers and restores those copies once the data has reached the destination. Delta is able to identify duplicated information and determine the most space efficient way to represent it. Initial tests show about 50% reduction in data movement while maintaining the same data quality and transmission frequency.

Contents

List of Figures

Chapter 1

Introduction

Scientific simulation workflows are becoming increasingly complex as the data size and the computation speed accelerate. With disk-based storage array performance falling behind these trends and faster alternatives, such as SSDs, still too expensive as a general solution, a move to online Integrated Application Workflows (IAWs) is starting. An illustration of such an IAW is in Figure 1.1. The general idea for the staging area, or burst buffer as they are sometimes called, is to offer a place to temporarily store intermediate data between IAW components. Data analytics or visualization components can retrieve this data both when it becomes available and when they have capacity. Current trends in extreme scale OS design suggest that this is better thought of as a logical rather than physical model as node partitioning is expected to offer a loosely coupled, nearly in-place data movement environment. This offers a degree of asynchrony while keeping data movement at interconnect speeds, or faster, rather than being limited by the storage array bandwidth. While shifting data movement online reduces the IO bottleneck, it is not a panacea.

The amount of relevant data produced during the run of such an application is large and is getting larger as the simulation ensembles use finer resolutions and more complex physics. Transferring such a high data volume repeatedly during runtime can have a severely negative impact on the network, where the improvement in bandwidth cannot match the growing data size, or on data storage due to a performance mismatch between data generators and consumers. The resulting backlog could affect performance on both the compute area and the staging area. Compounding the issue is the amount of energy consumed over the course of such an application including the cost of transferring data. Currently, computation is the single largest contributor to energy usage [5] but if the data volume grows at the same rate, the cost of moving data can easily become the limiting factor. Not only is writing to a storage array problematic, but also is moving data to other places within the compute resource. This energy cap prompts new work into managing various energy use sources throughout the scientific simulation workflow process. This paper examines one such project focusing on managing data movement.

For at least two different scientific simulation application classes, there are opportunities to reduce data movement with little to no negative IAW impact. For molecular dynamics and finite element codes, the data proportion that changes with each calculation iteration is significantly smaller than the entire working set size. Our measurements showed that only 40% to 75% of the data changed from iteration to iteration. Usually, the same type of data

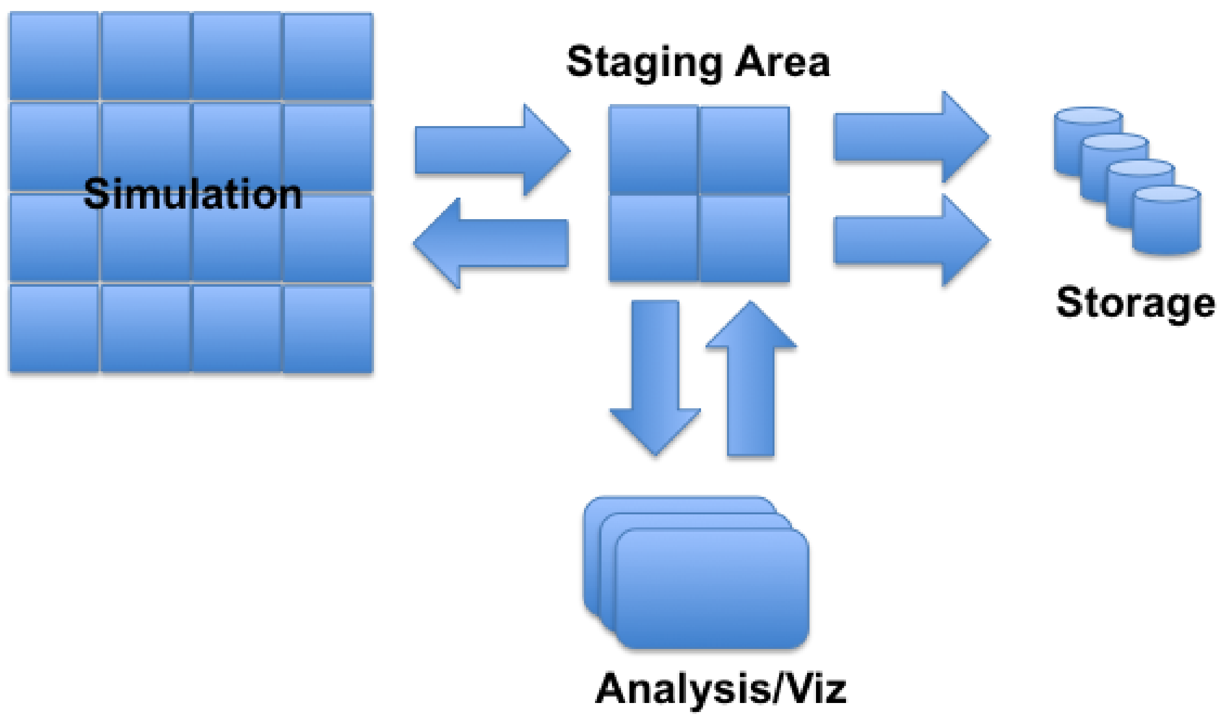


Figure 1.1. Overall IAW Architecture

is produced every timestep (e.g., temperature, velocity, position), but the value may or may not change from timestep to timestep offering opportunities to avoid data movement and the associated energy costs.

Using this observation, an application developer could adjust the application such that if a particular element does not change between timesteps, it is not sent to any downstream consumer. When the analysis software receives a "no change" or even no value at all instead of the element, it can assume that it has not changed from the previous value and use the old value instead. An even better and easier approach for application developers would be for an underlying system that did this for any application running on top of it. We developed such a system to effectively eliminate any data unchanged between output steps and yielded approximately a 50% reduction in aggregate data movement over the simulation lifetime.

The rest of the paper is organized as follows. First is a short discussion of related work in Section 2. Next is a design overview in Section 3. An evaluation follows in Section 4. Section 5 has a short discussion of the evaluation implications and future work.

Chapter 2

Related Work

General workflow systems such as Pegasus [10], Kepler [8], and DAGMan [9] manage the inter-component scheduling for an effective workflow. In general, they assume that each component is a separate application and typically disk is used to store intermediate data. These facts makes these systems less useful for constructing an online workflow system. Hand-coded systems using a language like Python generally suffer the same limitations.

Data storage solutions like Zookeeper [4] are infeasible for this environment as well. The inherent assumption of a single server containing a full copy of all of the data cannot work for this environment. If a multi-TB output is pushed into a staging area, there must be sufficient storage across a sufficient number of nodes to handle the data load. The inherent data replication is also undesirable. Further, each record is treated as a separate or full replacement entity. While Delta's concepts could be incorporated into a system like Zookeeper, it would require more semantic changes because an update can come from any source eliminating the ability to effectively determine on the client side what data has changed since the last update.

More directly related work includes the ConCORD [14] project. This work observed that data in memory could be better thought of based on the accesses made against it affording opportunities to reduce data duplication. While this works well in the target virtual machine environment, it does not address data moving off a node.

AI-Ckpt [11] evaluated memory changes for scientific simulations to only move data pages that change between output steps. While this is an admirable first step in this direction, our observations showed that far greater data movement reductions could be achieved by making decisions on an element-by-element basis because portions of every simulation variable generally changed every iteration while certain portions did not. With these static and dynamic elements intermixed, data pages are generally all moved even though only a portion of the data has changed.

On a post-processing basis, a climate science team evaluated lossy data compression techniques to determine how much a data set could be compressed while still maintaining sufficient validity to keep application scientists confident [1]. This project yielded good results, but is intended for long-term archival rather than the immediacy required by IAWs. The additional steps for data compression may be appropriate as an alternative to this approach, but would require considerably more computation, but at a lesser local storage cost. Further investigation would be required to determine which solution is preferable for which situations and if this globally optimized compression could be done on a node-by-node basis.

Chapter 3

Design and Implementation

One of the simplest way to examine the potential impact for this approach is to adapt an existing IO library to automatically handle both the data compression and expansion. The easiest library in which to add these sorts of data processing techniques is ADIOS [6]. The ability to install a new transport method that receives raw data with full name and type information makes it easy to replace any IO operation with data processing and then IO. The HDF5 Virtual Object Layer (VOL) offers similar functionality, but with potentially more rigid programming semantics.

Delta is a prototype ADIOS transport method that caches the last whole data set transferred for each output group and creates and expands compressed data sets only transferring the reduced data between the source and destination. As far as the end user is concerned, the IAW writes and reads data normally, but using the Delta transport method instead. The architecture is illustrated in Figure 3.1.

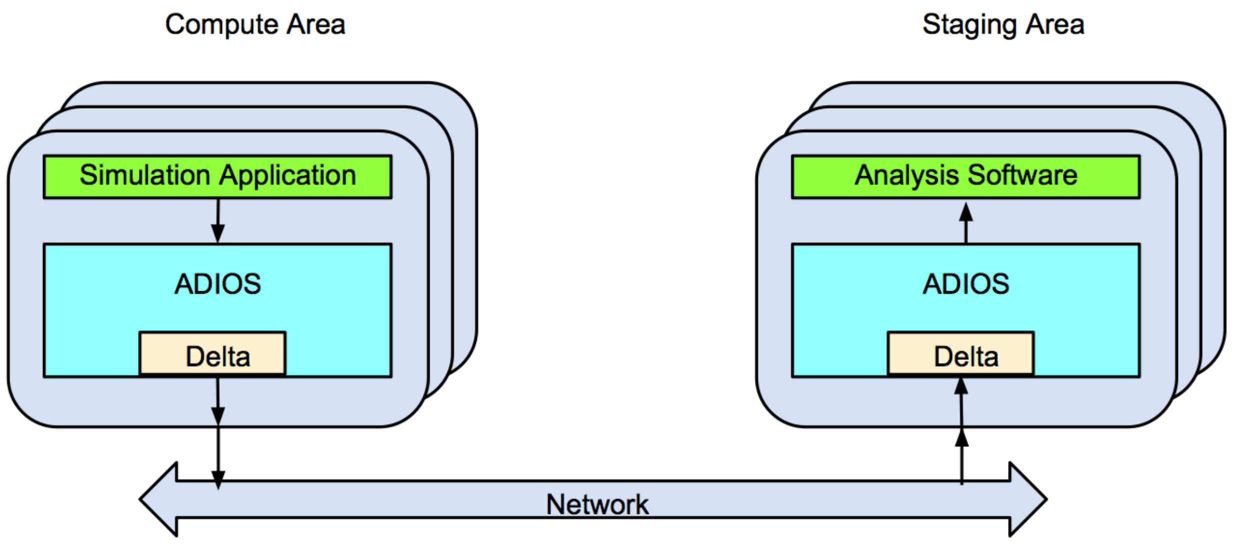


Figure 3.1. A diagram of the IAW architecture, including ADIOS and Delta

In order to reduce data transferred over the network, Delta must determine how much data is changed between every computation output. Each node in the system keeps track of the local full output from the previous full output from which it generates the difference. When the current round completes, the new output is compared to the cached full output. During the first round, there is no comparison because there is no old output with which to compare. Every subsequent round compares each variable element against its matching predecessor including both scalars and vectors. Delta calculates the difference between the rounds. Anything that has not changed is not included in the new payload and is instead replaced by metadata to describe what is left out. Once the payload arrives at its destination, the full, current data set can be recreated. The unchanged elements can be found in a previous payload. If the number of changed elements is too large, then the cost of the metadata summed with the changed data will be greater than the cost of just sending the data plainly. Therefore, if the amount of change is above a certain threshold, all the data is sent as is.

Packing the data for transport requires several steps. At top of each payload, certain pieces of information are required. Some of these data items are artifacts of how ADIOS encodes data. For example, because ADIOS uses a log-based format by default that annotates every log entry with information about the source process rank, this additional data is required. Using HDF5 VOL instead would eliminate some of these values, but would require more communication to coordinate among the participating processes to determine what has changed with each output step.

- Group ID: The identifier for the ADIOS group designated in the adios_open() call. In ADIOS, a group is a set of variables that are transported with the same transport method. A group can consist of both scalar and vector variables, each with different data types and sizes. A single application can consist of many groups, each possibly using a different transport method.
- Epoch: The epoch is the identifier of the current computation round. This value is written once per payload, regardless of the number of variables.
- Current Rank: The rank of the process preparing to write the current payload. This is written once per payload.
- Group Name Length: The reader will need to know the length of the name of the current group to read it properly.
- Group Name: The name of the group involved in the current payload is required. This value is only written once throughout the entire computation as well as its length. After this, the group id will be sufficient to identify the group.
- Variable Count: The number of variables belonging to the group that was designated during the adios_open() call. This is written only once during the length of the operation.

The previously mentioned parameters are written once during each adios_open-adios_close cycle except for the variable count of a previously written group. Variable counts will not change during the course of a full operation. After this section, the individual variables for the designated group are added. Each variable has two required components:

- Status: The status has a value of NONE, SOME or NEW. If the status is SOME, there was a change in the variable between the rounds. If the status is NONE, there was no change and no data should be sent besides the required metadata. If the status is NEW, that means that the variable in question is being written for the first time and all the necessary metadata will be sent along with it.
- Id: The variable id number used to identify the variable once it is delivered by the receiver.

  If the status is NONE, then only these two fields are sent as metadata for a particular variable. The actual data is exactly the same as the previous round. If the status is SOME, then more data is required, but since this variable was previously written, some pieces of information can be left out. Otherwise, the variable is new and will be padded accordingly.
- Name Length: The length of the variable name, needed to read in the name when the variable is being unpacked. This is only included for new variables being written for the first time.
- Name: The name of the variable, used to identify the variable in the user application. This is also only required for first time variables. During subsequent rounds, the Id will be a sufficient identifier for the variable.
- Dim Count: The number of dimensions for the variable. If the variable is a scalar, this value is 0.
- Global: A Boolean value that signals whether or not the variable is global. A global array has its dimensions and values split among the processes involved in the computation. The way the array is divided is user defined and it is up to the reader method to correctly identify an individual piece should it be requested.
- Dim Sizes: If the variable is a vector, then the size of each dimension must be included in the metadata. If the vector is global, then for each dimension, three pieces of information are necessary. First, the global value represents the total size of the dimension across every involved process. Second, the local value represents the starting point in the current dimension for the current process. Finally, the offset represents the amount of entries in the dimension that belong to the current process, starting from the local value.
- Type: This is an integer that represents the data type. The type only needs to be written the first time the variable is encountered.
- Type Size: This is the size of the specified data type. This information is also only required the first time.

Next, it must be determined whether all of the data should be sent or if a portion should be left out with some metadata that describes it. If there are no copies of the variable, then all of the information is sent. If the variable is a scalar, this is not an issue. If a scalar changed, it must be sent. If it did not change, it is not sent. Vectors present a challenge. Delta represents a vector variable using a bit vector. The bit vector is set to the size of the full output from the round. For example, if the variable is question was an array with 100 elements, the bit vector would consist of 100 bits. Each element in the output is compared to an element in the same position in the previous round. If it is the same as before, the value of the matching bit is set to 0 and the element is excluded from the payload. Otherwise, the bit is set to 1 and the data remains in the payload. If the new output vector is larger than the last, then the overflow is all represented by ‘1’s. All of the extra new data is included in the payload. The output from the last round is then replaced by the current output. Next, the size of the current rounds output is compared to the size of the reduced payload plus the bit vector. If the reduction summed with the bit vector is larger than the original output size, then the bit vector is discarded and all of the data is sent. Otherwise, the reduction and the bit vector are prepared for transport. Here are the remaining metadata fields:

- Total Size: The total size of the payload data (number of elements multiplied by the type size).
- Data: The actual payload whether it is the full output or the reduced output.
- Bit Count: The number of bits in the bit vector. If this is 0, then there is no bit vector, meaning that it was not worth it to reduce the output or all of the data was changed from the last round (or, as a special case, it is the first round). The bit count also tells the size of the original output, for unpacking purposes.
- Bit Vector: The actual bit vector. If the bit count was 0, then the bit vector is not included in the metadata.

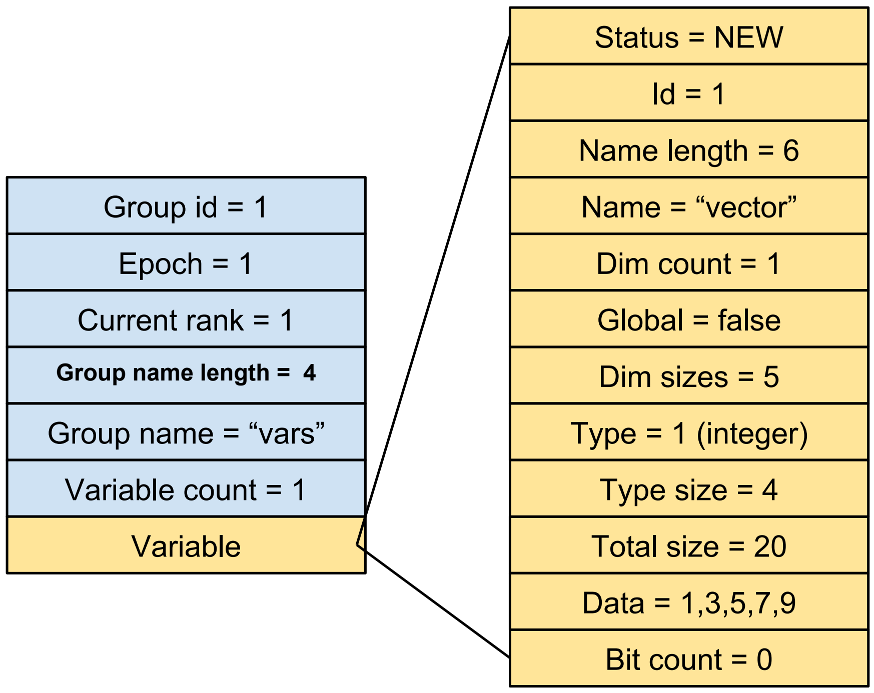
This is the format for every variable in every output group. When complete, the buffer is written to whatever the destination is. Figure 3.2(a) illustrates a full data set with all of the associated metadata included. Figure 3.2(b) shows the same data set in the following epoch. Some of its data values have changed, so a bit vector has been added to describe the change. Much of the metadata is no longer necessary at that point since it is unchanging and has already been written previously.

In order to unpack the payload, there are a set of data structures to manage the parts of the payload. At the top is the Delta_Data_Struct, which keeps track of two different types of information: The information that can change from epoch to epoch and the information that stays the same.

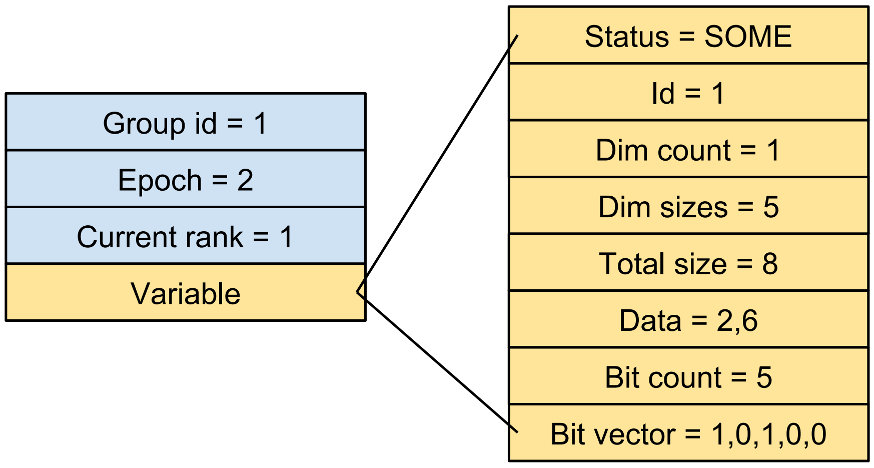
For the changing information, The Delta_Data_Struct (or DDS) keeps a linked list of Group_Struct structures. Each Group_Struct represents a different group and contains a linked list of Epoch structures. An Epoch structure represents a single timestep in the associated group’s lifespan. It also keeps track of the process (rank) that submitted the group data during that timestep. Finally, each Epoch has a vector of Var_Struct structures that hold the state of each variable during that particular epoch such as its value and size (if it is a vector, for example).

For the unchanging information, the DDS has a linked list of Group_Record structures that hold the group name and the number of variables for that group since those don’t change. Also, each contains a vector of Variable_Record structures that hold information such as the variable name, data type, and whether or not it is global.

The data could be stored as the difference or reconstituted into the full data set. For this first test case, we simply regenerated the whole data set each time removing complexity from read operations.



(a) Full Payload



(b) Difference Payload

Figure 3.2. Payload Descriptions

Chapter 4

Evaluation

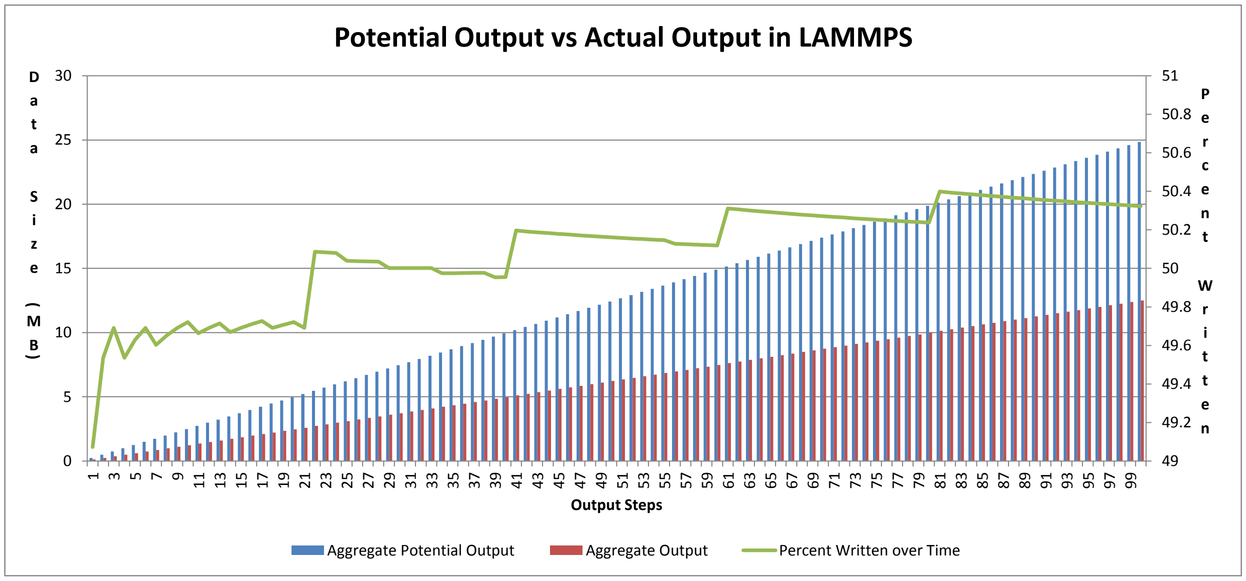


Figure 4.1. A graph showing the pattern of reduced writes in LAMMPS for the "Crack" example

The evaluation is performed on the Chama capacity cluster at Sandia National Laboratories. It consists of 1232 nodes each with 2 2.6 GHz Intel Sandy-Bridge CPUs with 8 cores/socket. Each node has 64 GB of DDR3 RAM and connects with a 4X QDR InfiniBand network configured in a fat tree. The file system is the site shared Lustre offering 1 PB of storage. There are also 8 login nodes each with the same hardware as the compute nodes. All nodes run RHEL 6.

To evaluate this approach's potential, two separate scientific applications were modified to measure the change of output between timesteps. The first is the molecular dynamics code LAMMPS [13]. It can simulate a number of interactions and events at a fine level by applying physics to atom positions based on the simulation setup. The user can define the interaction in an application specific manner and let it run over a number of timesteps. Here, the modified code was run with and example called "crack", which simulates a crack propagating through some solid. Roughly 40% of the data changed between output timesteps, getting as high as 60% at times. Figure 4.1 displays the changes aggregated over time. Another example, which simulated a melting solid had 60% of its variables changing between output timesteps, getting as high as 75%. Similar results were found when running an example built using the DEAL.II [2] finite element library. It is easy to see that this strategy will work better with some simulations that others. Simulations that have some form of propagation (such as a crack or melting) would have many of its parts remain static for some period of time. Other simulations even have parts that don't change at all (such as the ground over which a liquid is flowing). These kinds of simulations would benefit the most from applying Delta or an application specific version. Since ADIOS can be used to support such applications, Delta aims to be an application independent version.

Chapter 5

Conclusion

As supercomputers continue gaining ground on the road to exascale, many of the currently acceptable practices when it comes to managing data will become obsolete because of the resulting bottlenecks and the associated energy consumption. One of these practices involves taking up bandwidth to transfer information that was already previously transmitted and already exists at the destination. Delta makes an attempt to prevent that by keeping track of changes between timesteps in a computation in order to detect stagnant data. By blocking the transfer of such data, the extra bandwidth that would have been consumed is now available to the rest of the application or not used at all. In that process, no data is lost to the staging area for IAWs. The reader in Delta can rebuild the reduced data using previously obtained information, giving a complete picture of the output produced during computation.

There are many avenues to continue this work. First, ADIOS has an inherent assumption that the variables represented in an output can change. The current design does not take this into account. There are also considerations related to static metadata. For example, global or local array dimensions for structure meshes may not change over the simulation lifetime. Retransmitting these values each time could be eliminated further reducing data sizes.

Different data encoding techniques, such as using a sparse map when few elements change could also be incorporated along with a flag identifying which encoding technique is employed.

Maintaining a fixed "full" data copy is sufficient for a prototype, but not an optimal solution. Ideally, this "full" data copy would update with each output step reducing the frequency of full data set transfers.

Other higher computational cost techniques such as lossless or lossy data compression could be used instead. The performance, space, and energy tradeoffs must be investigated to see when these more complex approaches would be superior to this simple approach.

References

[1] Allison H. Baker, Haiying Xu, John M. Dennis, Michael N. Levy, Doug Nychka, Sheri A. Mickelson, Jim Edwards, Mariana Vertenstein, and Al Wegener. A methodology for evaluating the impact of data compression on climate simulation data. In *The 23rd International Symposium on High-Performance Parallel*, pages 203–214, 2014.

[2] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II – a general purpose object oriented finite element library. *ACM Trans. Math. Softw.*, 33(4):24/1–24/27, 2007.

[3] Jai Dayal, Jay Lofstead, Karsten Schwan, and Ron Oldfield. Resilient data staging through mxn distributed transactions. Supercomputing 2011 Parallel Data Storage Workshop poster, 2011.

[4] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *In USENIX Annual Technical Conference*, 2010.

[5] James H Laros III, Kevin T Pedretti, Suzanne M Kelly, Wei Shu, and Courtenay T Vaughan. Energy based performance tuning for large scale high performance computing systems. In *Proceedings of the 2012 Symposium on High Performance Computing*, page 6. Society for Computer Simulation International, 2012.

[6] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich IO methods for portable high performance IO. [7].

[7] Jay Lofstead, Fang Zheng, Scott Klasky, and Karsten Schwan. Adaptable, metadata rich IO methods for portable high performance IO. Rome, Italy, 2009.

[8] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

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

[10] Sape J. Mullender, Ian M. Leslie, and Derek McAuley. Operating-system support for distributed multimedia. pages 209–219, 1994.

[11] Bogdan Nicolae and Franck Cappello. Ai-ckpt: leveraging memory access patterns for adaptive asynchronous incremental checkpointing. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 155–166. ACM, 2013.

[12] Beth Plale, Matei Ripeanu, Franck Cappello, and Dongyan Xu, editors. *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014*. ACM, 2014.

[13] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[14] Lei Xia, Kyle C. Hale, and Peter A. Dinda. Concord: easily exploiting memory content redundancy through the content-aware service command. In *The 23rd International Symposium on High-Performance Parallel*, pages 25–36, 2014.

DISTRIBUTION:

1	MS	1319	Ron Oldfield, 1461
1	MS	0899	Technical Library, 9536 (electronic copy)

v1.40

Sandia National Laboratories