Data Intensive Computing for Bioinformatics

Judy Qiu[1], Jaliya Ekanayake[1,2], Thilina Gunarathne[1,2], Jong Youl Choi[1,2], Seung-Hee Bae[1,2], Yang Ruan[1,2], Saliya Ekanayake[1,2], Stephen Wu[1,2], Scott Beason[1], Geoffrey Fox[1,2], Mina Rho[2], Haixu Tang[2]

[1]Pervasive Technology Institute, [2]School of Informatics and Computing
Indiana University
Bloomington IN, U.S.A.
{xqiu, jekanaya, tgunarat, jychoi, sebae, yangruan, sekanaya, taklwu, smbeason, gcf, mrho, hatang@indiana.edu}

1 INTRODUCTION

1.1 Overview

Data intensive computing, cloud computing, and multicore computing are converging as frontiers to address massive data problems with hybrid programming models and/or runtimes including MapReduce, MPI, and parallel threading on multicore platforms. A major challenge is to utilize these technologies and large scale computing resources effectively to advance fundamental science discoveries such as those in Life Sciences. The recently developed next-generation sequencers have enabled large-scale genome sequencing in areas such as environmental sample sequencing leading to metagenomic studies of collections of genes. Metagenomic research is just one of the areas that present a significant computational challenge because of the amount and complexity of data to be processed.

This chapter builds on research we have performed (Ekanayake, Gunarathne, & Qiu, Cloud Technologies for Bioinformatics Applications, 2010) (Ekanayake J. , et al., 2009) (Ekanayake, Pallickara, & Fox, MapReduce for Data Intensive Scientific Analyses, 2008) (Fox, et al., 2009) (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008) (Qiu, et al., 2009) (Qiu & Fox, Data Mining on Multicore Clusters, 2008) (Qiu X. , Fox, Yuan, Bae, Chrysanthakopoulos, & Nielsen, 2008) (i-MapReduce, 2009) on the use of Dryad (Microsoft's MapReduce) (Isard, Budiu, Yu, Birrell, & Fetterly, 2007) and Hadoop (open source) (Apache Hadoop, 2009) to address problems in several areas, such as particle physics and biology. The latter often have the striking all pairs (or doubly data parallel) structure highlighted by Thain (Moretti, Bui, Hollingsworth, Rich, Flynn, & Thain, 2009). We discuss here, work on new algorithms in section 2, and new programming models in sections 3 and 4.

We have a robust parallel Dimension Reduction and Deterministic Annealing clustering, and a matching visualization package. We also have parallel implementations of two major dimension reduction algorithms – the SMACOF approach to MDS and Generative Topographic Mapping (GTM) described in section 2. MDS is $O(N^2)$ and GTM O(N) but, since GTM requires the points to have (high dimensional) vectors associated with them, only MDS can be applied to most sequences. Also, since simultaneous multiple sequence alignment MSA is impractical for interesting biological datasets, MDS is a better approach to dimension reduction for sequence samples, because it only requires sequences to be independently aligned in pairs to calculate their dissimilarities. On the other hand, GTM is attractive for analyzing high dimension data base records, where well defined vectors are associated with each point – in our case each database record. Distance calculations (Smith-Waterman-Gotoh) MDS and clustering are all $O(N^2)$, and will not properly scale to multi-million sequence problems and hierarchical operations to

address this are currently not supported for MDS and clustering except in a clumsy manual fashion. In the final part of section 2, we propose a new multiscale (hierarchical) approach to MDS that could reduce complexity from $O(N^2)$ to O(NlogN) using ideas related to approaches already well understood in $O(N^2)$ particle dynamics problems.

In sections 3 and 4, we chose to focus on the MapReduce frameworks, as these stem from the commercial information retrieval field, which is perhaps currently the world's most demanding data analysis problem. Exploiting commercial approaches offers a good chance that one can achieve high-quality, robust environments, and MapReduce has a mixture of commercial and open source implementations. In particular, we have looked at MapReduce and MPI, and shown how to analyze biological samples with modest numbers of sequence on a modern 768 core 32 node cluster. We have learnt that currently MapReduce cannot efficiently perform clustering and MDS (Multidimensional Scaling) steps, even though the corresponding MPI implementation only needs reduction and broadcast operations and so fit architecturally functions supported in MapReduce. In addition, since we need to support iterative operations, we propose the use of a modified MapReduce framework called i-*MapReduce*. An early prototype described in section 4 has been run on kernels but, as of this time, not on complete bioinformatics applications. Research issues include fault tolerance, performance, and support for existing MPI programs with the i-*MapReduce* run time supporting the subset of MPI calls. We also demonstrate in section 3 how we take "all-pairs" or "doubly data parallel" computations in two important bioinformatics sequencing applications, and use the results to compare two implementations of MapReduce (Dryad and Hadoop) with MPI. We describe an interesting technology developed to support rapid changes of operating environment of our clusters. We focus on the effects of inhomogeneous data and set the scene for discussion of i-MapReduce in section 4. One of the biological applications – sequence assembly by Cap3 – is purely "Map" and has no reduction operation. The other – calculation of Smith-Waterman dissimilarities for sequences – has a significant reduction phase to concentrate data for later MDS and Clustering.

1.2 Architecture for Data Intensive Biology Sequence Studies

The data deluge continues throughout science, and practically all scientific areas need analysis pipelines or workflows to propel the data from the instrument through various stages to scientific discovery, often aided by visualization. It is well known that these pipelines typically offer natural data parallelism that can be implemented within many different frameworks.

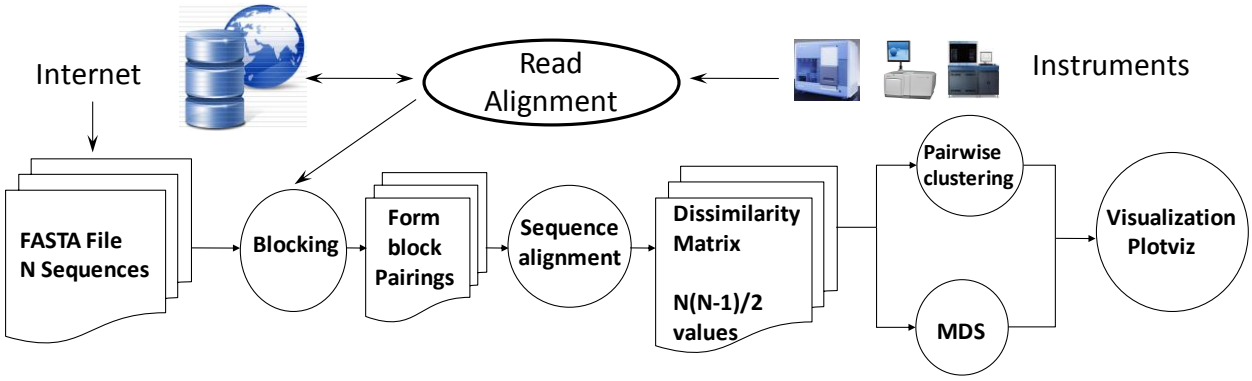


Figure 1. Pipeline for analysis of metagenomics Data

Figure 1 shows the data analysis pipeline shared by many gene sequence studies and, in particular, by our early work on metagenomics. Apart from simple data manipulation, there are three major steps – calculation of the pairwise distances between sequences, followed by MDS, and Clustering. We focus on the former here as it can use current MapReduce technologies, and exhibits a doubly data parallel structure, since the dissimilarities δ_{ij} can be calculated independently for the N distinct labels of sequences *i* and *j*. Note that, currently, one cannot reliably use multiple sequence analysis (MSA) on large samples, which means techniques that only use pairwise distances between sequences (that can be reliably calculated) must be used, instead of methods relying on vector representations of the sequences. The lack of vector representation for sequences implies that many approaches to dimension reduction (such as GTM (Bishop & Svensén, GTM: A principled alternative to the self-organizing map, 1997)) and clustering (such as original vector-based Deterministic annealing clustering (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998)) cannot be used. We have published several papers (Fox, et al., 2009) (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008) (Qiu, et al., 2009) describing our earlier work on this pipeline and its related problems. The pairwise distances for metagenomics and other gene family problems are calculated using the algorithm developed by Smith-Waterman (Smith & Waterman, 1981) and Gotoh (Gotoh, 1982) (SW-G), but the process is complicated by the need to exploit the symmetry $\delta_{ij} = \delta_{ji}$, and to arrange the results in a form suitable for the next steps in the pipeline. We have obtained detailed performance measurements on MPI, Hadoop and Dryad with results summarized in section 3. This section also describes work on CAP3, which only involves the initial doubly data parallel read alignment stage.

In section 2, we use data from the NIH database PubChem (Wheeler, et al., 2006) (PubChem project, 2009) that records properties of chemical compounds. Currently there are 26 million compounds, but in our initial studies we use random subsets of up to 100,000 compounds. We then use 166 binary properties to define the 166 dimensional vectors associated with each compound. In a follow up work we are using interpolation and other methods to extend analysis to the entire NIH dataset.

2 INNOVATIONS IN ALGORITHMS FOR DATA INTENSIVE COMPUTING

2.1 Visualization analysis by using parallel MDS and GTM

Dimension reduction and follow-up visualization of large and high-dimensional data in low dimensions is a task of growing importance in many fields of data mining and information retrieval to understand data structures, verify the results of data mining approaches, or browse them in a way that distance between points in visualization space (typically 2D or 3D) tracks the one in original high dimensional space. There are several well understood approaches to dimension reduction, but they can be very time and memory intensive for large problems. In this section we discuss parallel algorithms for Scaling by MAjorizing a COmplicated Function (SMACOF) to solve Multidimensional Scaling (MDS) problems, and Generative Topographic Mapping (GTM). The former is particularly time consuming, with complexity that grows as square of data set size. However, it does have the advantage of not requiring explicit vectors for dataset points, but only the measurement of inter-point dissimilarities. We also present a comparison between MDS and GTM by using Canonical Correlation Analysis (CCA).

Multidimensional Scaling (MDS): MDS (Kruskal & Wish, 1978), (de Leeuw, Applications of convex analysis to multidimensional scaling, 1977), (de Leeuw, Convergence of the majorization method for multidimensional scaling, 1988), (Borg & Groenen, 2005) is a technique for mapping generally high-dimensional data into a target dimension (typically a low dimension L), such that each distance between a pair of points in the mapped configuration is an approximation to the corresponding given pairwise

proximity value as measured by a weighted least squares sum. The given proximity information is represented as an $N \times N$ dissimilarity matrix ($\Delta = [\delta_{ij}], 1 \leq i, j \leq N$), where N is the number of points (objects), and δ_{ij} is the dissimilarity between point i and j. The output of MDS algorithms can be represented as an $N \times L$ configuration matrix X, whose rows represent each data points x_i ($i = 1, \dots, N$) in L-dimensional space. We are able to evaluate how well the given points are configured in the L-dimensional space by using a least squares style objective functions for MDS, called STRESS (Kruskal J. , 1964) or SSTRESS (Takane, Young, & de Leeuw, 1977). Definitions of STRESS (2.1) and SSTRESS (2.2) are given in the following equations:

$$\sigma(X) = \sum_{i<j \leq N} w_{ij} \left(d_{ij}(X) - \delta_{ij}\right)^2 \quad (2.1)$$

$$\sigma^2(X) = \sum_{i<j \leq N} w_{ij} \left(d_{ij}^2(X) - \delta_{ij}^2\right)^2 \quad (2.2)$$

where $d_{ij}(X) = ||x_i - x_j||$, $1 \leq i < j \leq N$ in the L-dimensional target space, and w_{ij} is a weight value, with $w_{ij} \geq 0$.

Generative Topographic Mapping (GTM): GTM is an unsupervised learning algorithm for modeling the probability density of data, and finding a non-linear mapping of high-dimensional data in a low-dimension space. GTM is also known as a principled alternative to Self-Organizing Map (SOM) (Kohonen, 1998), which does not have any density model. GTM defines an explicit probability density model based on Gaussian distribution (Bishop & Svensén, GTM: A principled alternative to the self-organizing map, 1997) and seeks the best set of parameters associated with Gaussian mixtures by using an optimization method, notably the Expectation-Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977).

Canonical Correlation Analysis (CCA): CCA is a classical statistical method to measure correlations between two sets of variables in their linear relationships (Hotelling, 1936). Contrary to ordinary correlation measurement methods, CCA has the ability to measure correlations of multidimensional datasets by finding an optimal projection to maximize the correlation in the subspace spanned by features. The projected values, also known as *canonical correlation variables*, can show how two input sets are correlated. In our experiments, we have measured the similarity of MDS and GTM results by measuring correlation in CCA. More details of CCA can be found in (Hardoon, Szedmak, & Shawe-Taylor, 2004) (Campbell & Atchley, 1981) (Thompson, 1984).

2.1.1 Parallel MDS and GTM

Running MDS or GTM with large datasets (such as PubChem) requires memory-bounded computation, and are not necessarily CPU-bounded. For example, GTM may need a matrix for 8,000 latent points, corresponding to a 20x20x20 3D grid, with 100,000 data points, which requires at least 6.4 GB memory space for holding 8-byte double precision numbers. This single requirement easily prevents us from using a single process to run GTM. Also, memory requirement of SMACOF algorithm increases quadratically as N increases. For example, if $N = 100{,}000$, then one $N \times N$ matrix needs 80 GB of memory to hold 8-byte double precision numbers. To make matters worse, the SMACOF algorithm generally needs six $N \times N$ matrices, which means at least 480 GB of memory is required to run SMACOF with 100,000 data points (excluding other memory requirements). To overcome this problem, we have developed parallel

MDS (SMACOF) and GTM algorithms by using MPI Message Passing Interface (MPI, 2009), which we now discuss in more detail.

Parallel SMACOF: Scaling by MAjorizing a COmplicated Function (SMACOF) (de Leeuw, Applications of convex analysis to multidimensional scaling, 1977), is an algorithm to solve MDS problem with STRESS criterion based on an iterative majorization approach, where one iteration consists of two matrix multiplications. For the mathematical details of SMACOF algorithm, please refer to (Borg & Groenen, 2005). To parallelize SMACOF, we decompose each $N \times N$ matrix with a $m \times n$ block decomposition, where m is the number of block rows and n is the number of block columns, to make use of a total of $p(= m \times n)$ processes. Thus, each process requires only approximately a $1/p$ of the sequential memory requirement of SMACOF algorithm. Figure 2 illustrates how a matrix multiplication between an $N \times N$ matrix (M) and an $N \times L$ matrix (X) is done in parallel using MPI primitives when each $N \times N$ matrix (M) is decomposed with $p = 6, m = 2, n = 3$, and each arrow represents a message passing. For simplicity, we assume $N \bmod m = N \bmod n = 0$.

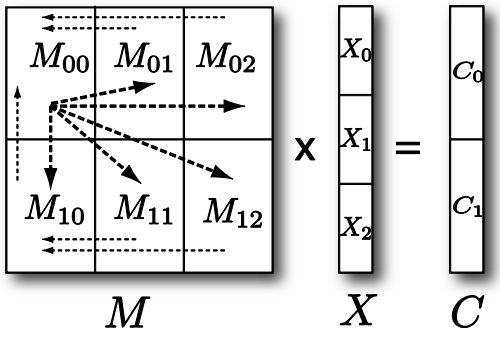


Figure 2. Parallel matrix multiplication of $N \times N$ matrix and $N \times L$ matrix based on the 2×3 block decomposition with 6 processes.

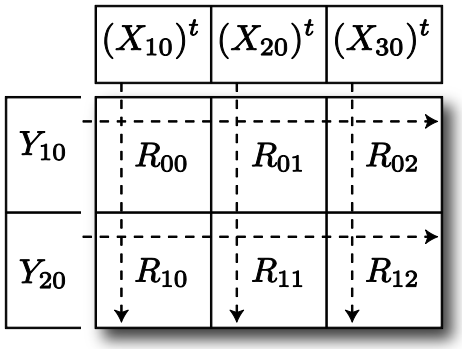


Figure 3. Data decomposition of parallel GTM for computing responsibility matrix R by using 2×3 mesh of 6 processes.

Parallel GTM: To develop the parallel GTM algorithm, we started by analyzing the original GTM algorithm. The GTM algorithm seeks a non-linear manifold embedding of user-defined K latent discrete variables y_k, mapped from a low L-dimension space called *latent space*, which can optimally represent the given N data points $x_n \in \mathbb{R}^D (n = 1, \dots, N)$ into the high D-dimension space, called *data space* (usually $L \ll D$). To define optimality, GTM uses the following log-likelihood function $\mathcal{L}$ using Gaussian noise model:

$$\mathcal{L} = \underset{\{y_k\},\beta}{\operatorname{argmax}} \sum_{n=1}^{N} \ln \left\{ \frac{1}{K} \sum_{k=1}^{K} \left(\frac{\beta}{2\pi} \right)^{D/2} \exp\left(-\frac{\beta}{2} \| x_n - y_k \|^2 \right) \right\} \qquad (2.3)$$

where β^{-1} represents the variance in Gaussian distribution. Since the detailed derivations of GTM algorithm is out of this section's scope, we recommend readers to refer to the original GTM papers (Bishop & Svensén, GTM: A principled alternative to the self-organizing map, 1997) (Bishop, Svensén, & Williams, GTM: The generative topographic mapping, 1998).

In GTM, the most memory consuming step for optimization is the process to compute the posterior probabilities, known as *responsibilities*, between K latent points and N data points (which is represented by a $K \times N$ matrix). The core of parallel GTM algorithm is to decompose the responsibility matrix into $m \times n$ sub-blocks (Figure 3). Each sub-block holds responsibilities for only approximately K/m mapped

point y_k's and N/n data point x_n's. Once the matrix is decomposed, each node of $m \times n$ mesh compute grids can process one sub-block, requiring only $1/mn$ of the memory spaces of the original full responsibility matrix.

2.1.2 Experimental Results

We have done a performance analysis of parallel MDS (SMACOF) and parallel GTM by using a 20K PubChem dataset having 166 dimensions, and measured the correlation of MDS and GTM results for a 100K PubChem dataset. For this performance measurement, we have used our modern cluster systems (Cluster-C, Cluster-E, and Cluster-F) as shown in Appendix.

Parallel MDS: Figure 4 shows the performance comparisons for 20K PubChem data with respect to decomposition methods for the $N \times N$ matrices with 32, 64, and 128 cores in Cluster-E and Cluster-C. A significant characteristic of those plots in Figure 4 is that skewed data decompositions (such as $p \times 1$ or $1 \times p$, which decompose by row-base or column-base), are always worse in performance than balanced data decompositions (such as $m \times n$ block decomposition which m and n are as similar as possible). There're several reasons for these results. One reason might be the cache line effect that affects cache reusability. In general, balanced block decompositions show better cache reusability (less cache misses) than the skewed ones (Bae, 2008) (Qiu & Fox, Data Mining on Multicore Clusters, 2008). Also, the difference of overhead in message passing mechanism for different data decompositions, specially for calculating $B(X)$, is another possibility for the results in the Figure 4.

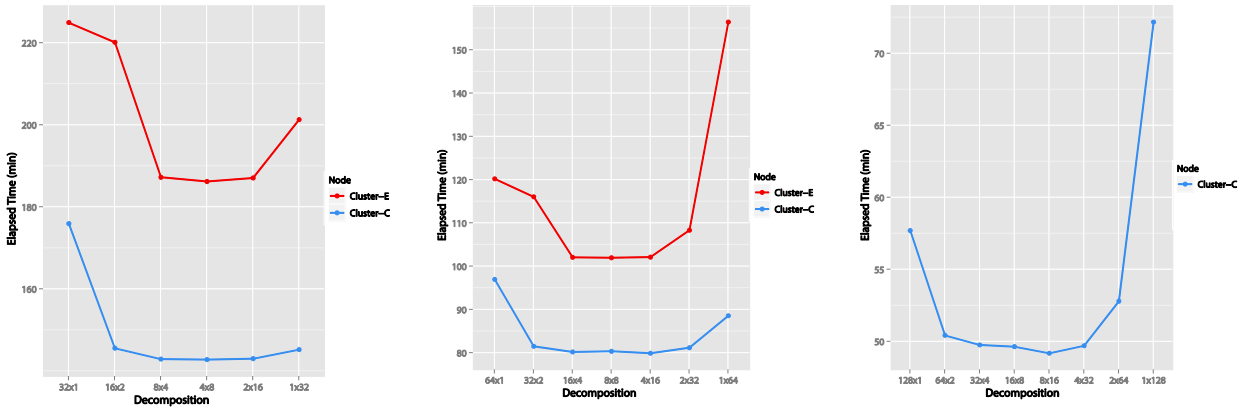
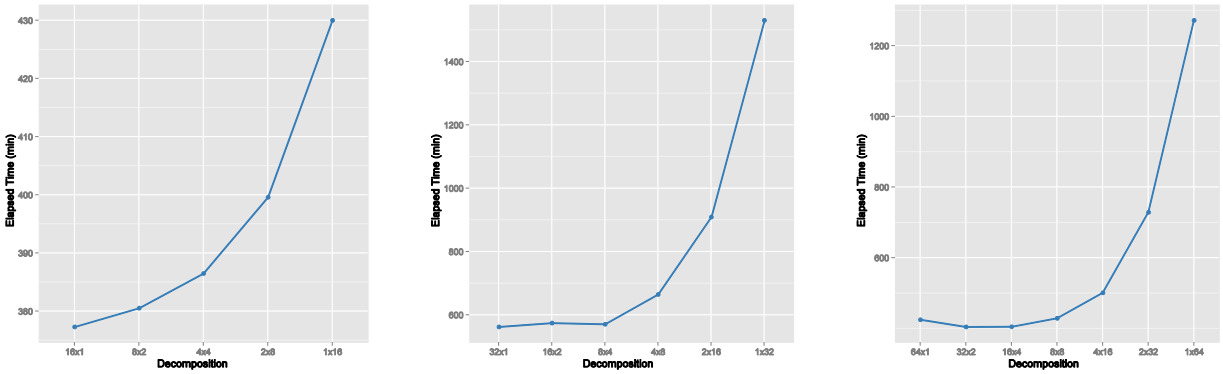


Figure 4. Performance of Parallel SMACOF for 20K PubChem data with 32,64, and 128 cores in Cluster-E and Cluster-C w.r.t. data decomposition of $N \times N$ matrices.

Parallel GTM: We have measured performance of parallel GTM with respect to each possible $m \times n$ decomposition of the responsibility matrix to use at most $p = mn$ cores for $p = 16$ (Cluster-F), plus 32 and 64 cores in Cluster-E and using the 20k PubChem dataset.

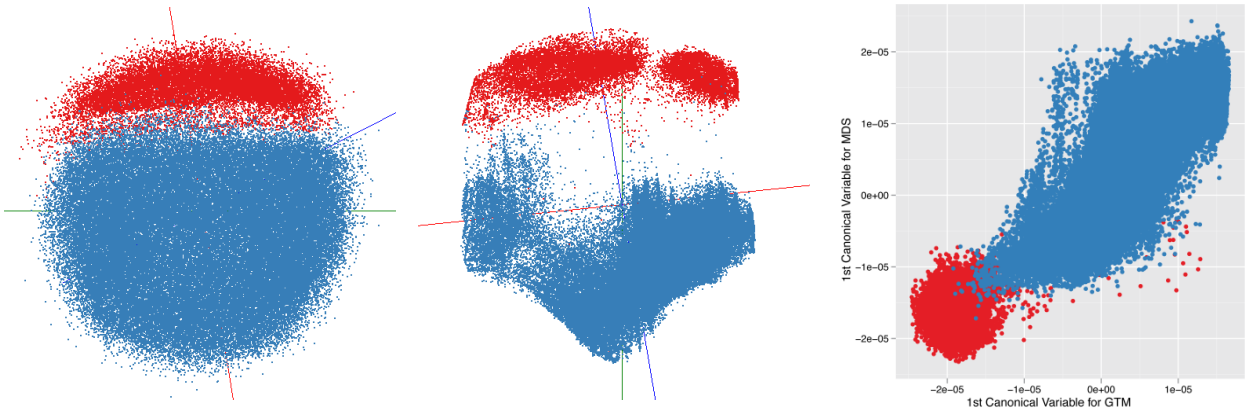


(a) 16 cores on Linux (Cluster-F) *(b) 32 cores on Windows (Cluster-E)* *(c) 64 cores on Windows (Cluster-E)*

Figure 5. Performance of Parallel GTM for 20K PubChem data with 16, 32 and 64 cores running on Cluster-E (32 and 64 cores) and Cluster-F (16 cores) plotted with absicca defining the the data decomposition running on $m \times n$ compute grids.

As shown in 0, the performance of parallel GTM is very sensitive on the choice of decomposition of responsible matrix R and, especially, the size of n affects greatly the performance. This is because the large n value increases the number of row-communications for exchanging sub-matrix of Y, while the submatrices of X doesn't need to re-distribute after starting processing since they are not changed throughout the whole process. Also, the results show that the worst case performance is not changed as much as we increase the number of cores. This implies that the worst performance is mainly due to the overheads caused by the use of MPI and its communications, not the process computing time in each core. The outperformance on Linux (0(a)) is because our parallel GTM implementation is using the statistics package R which is better optimized in Linux than Windows. In Windows (0(b) and (c)), we have obtained overall performance gains of about 16.89 (%) ~ 24.41 (%) by doubling the number of cores. Further current algorithm has an inherently sequential component. So we have succeeded in distributing the memory but we need further study of computational performance.

Correlation measurement by CCA: We have processed 100,000 PubChem data points by using our parallel MDS and GTM and measured similarity between MDS and GTM outputs by using CCA. As shown in Figure 6, the correlation shows a strong linear relationship between MDS and GTM outputs.



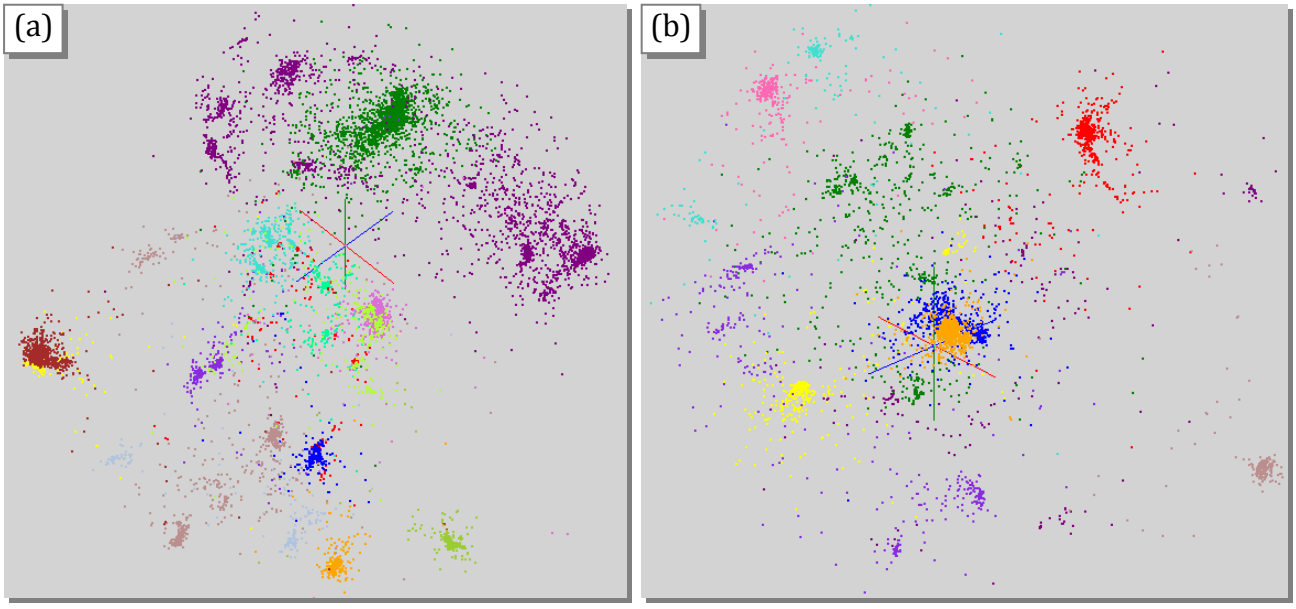
(a) MDS for 100K PubChem *(b) GTM for 100K PubChem* *(c) Canonical correlation variable plot for 100K PubChem MDS and GTM*

Figure 6. SMACOF and GTM outputs of 100K PubChem dataset are shown in (a) and (b). SMACOF and GTM correlation computed by CCA is shown in (c) as a plot with canonical correlation variables. In this result, the optimal correlation, so-called canonical correlation coefficient, is 0.90 (maximum is 1.00) which shows strong correlation between SMACOF and GTM.

2.1.3 Summary

In this sub-section, we have tried to deal with large data sets using parallelism in two different data mining algorithms (MDS(SMACOF) and GTM). We found that, for MDS, skewed data decompositions are always worse in performance than balanced data decompositions. In the GTM case, the choice of the responsible matrix R matters, and the worst performance is due to the overhead caused by MPI communication. However, there are important problems for which the data set size is too large for even our parallel algorithms to be practical. Because of this, we are now developing interpolation approaches for both algorithms. We are planning to run MDS or GTMs with a (random) subset of the dataset and the dimension reduction of the remaining points are interpolated, so that we can visualize many more data points without using huge amounts of memory.

2.2 Metagenomics Studies with Clustering



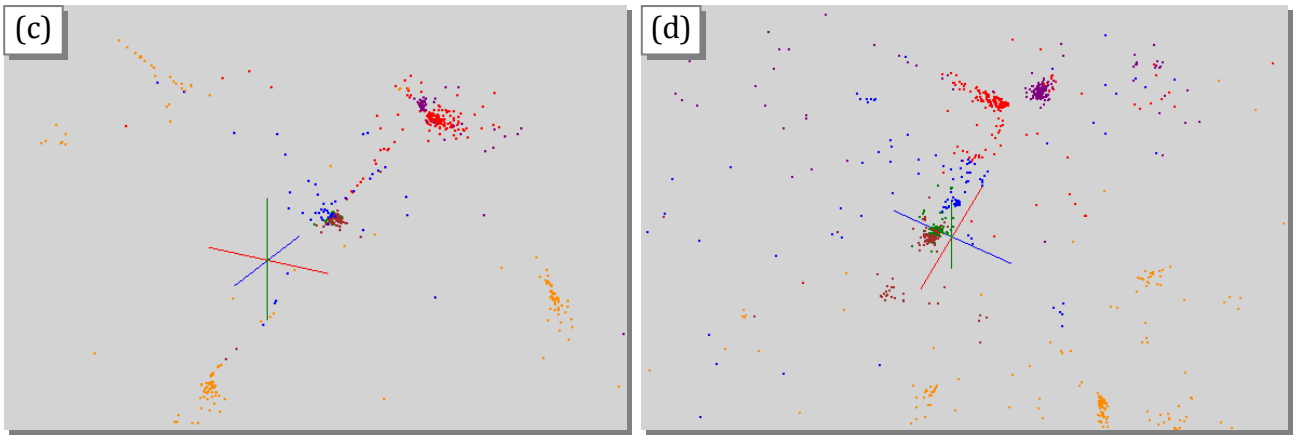


Figure 7. Results of Smith-Waterman distance Computation, Deterministic Annealing Clustering and MDS visualization pipeline for 30,000 Metagenomics sequences. (a) shows 17 clusters for full sample using Sammon's version of MDS for visualization. (b) shows 10 sub-clusters with a total of 9793 sequences found from purple and green clusters in (a) using Sammon's version of MDS for visualization.

Sub clustering of light brown cluster in Figure 7(a) with 2163 sequences decomposed further into 6 sub-clusters. In (c) Sammon's ansatz is used in MDS and in (d) SMACOF with less emphasis on small distances: weight(i,j) = 1 in equation (2.4).

Our initial work on Metagenomics builds on previous clustering work on clustering (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998) (Rose, Gurewitz, & Fox, A deterministic annealing approach to clustering, 1990) (Rose, Gurewitz, & Fox, 1990) (Hofmann & Buhmann, 1997) for determining families and dimension reduction using MDS for visualization (Klock & Buhmann, 2000) (Kearsley, Tapia, & Trosset, 1995) (Kruskal J. , 1964) (Takane, Young, & de Leeuw, 1977) (Kruskal & Wish, 1978) (Borg & Groenen, 2005) (de Leeuw, Applications of convex analysis to multidimensional scaling, 1977). We will propose in section 2.3 to research the use of MDS to reliably divide the sequence space into regions and support fast hierarchical algorithms. Typical results are shown in Figure 7 from an initial sample of 30,000 sequences.

Figure 7 illustrates clustering of a Metagenomics sample using the robust deterministic annealing approach described in (Rose, Gurewitz, & Fox, A deterministic annealing approach to clustering, 1990) (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998) (Hofmann & Buhmann, 1997) (Klock & Buhmann, 2000) (Qiu X. , Fox, Yuan, Bae, Chrysanthakopoulos, & Nielsen, 2008). This is implemented in MPI and runs in 30 minutes for 10 clusters and over 2 hours for 17 clusters on the 768 core cluster for the full 30,000 sequences. This processing time is proportional to the square of both the number of sequences and number of clusters. We need hierarchical methods to process large data samples and/or large number of clusters. This is illustrated for clusters in Figures Figure 7 (b, c, d). We will discuss more automatic approaches to this in section 2.3.

We can generalize equations (2.1) and (2.2) to state that MDS finds the best set of vectors $\underline{\mathbf{x}}_i$ in any chosen dimension d (d=3 in our case) by minimizing:

$$\sum_{i,j} w_{ij} \left({d_{ij}}^{n}(X) - {\delta_{ij}}^{m} \right)^2 \quad (2.4)$$

The form of the weights w_{ij} is chosen to reflect the importance of a point or perhaps a desire (Sammon's method with $w_{ij} = 1/\delta_{ij}$ as opposed to SMACOF with weight w_{ij} =1 to fit smaller distance more precisely than larger ones. The index *n* is typically 1 (Euclidean distance) but 2 is also useful. The index *m* is 1 in Figure 7 but *m*=0.5 is also interesting. Figure 7(c and d) show the sensitivity to MDS heuristic with SMACOF producing better results than Sammon for the sub-clustering of the smallish 2163 sequence sample. Generally we use Sammon since it gives the best results overall.

We have MDS implementations with three different methods – the classic expectation maximization approach (Kruskal & Wish, 1978) (Borg & Groenen, 2005) described in section 2.1, a deterministic annealing version (Klock & Buhmann, 2000) (Klock & Buhmann, 2000), and a distinct version that uses nonlinear χ^2 solution methods (Kearsley, Tapia, & Trosset, 1995) which was used in Figure 7. All have efficient parallel implementations (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008), and we will describe the second and third approaches in detail elsewhere.

Deterministic annealing (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998) is a powerful idea to avoid local minima in optimization methods (and both clustering and MDS can be considered this way). By calculating averages explicitly using mean field theory, we can simplify the notoriously slow simulated annealing. For clustering, Hofmann and Buhmann (Hofmann & Buhmann, 1997) first showed how to do this in a formulation that only uses pairwise distances. To see that, define an energy function

$$\mathrm{H_{PWDA}} = 0.5 \sum_{i=1}^{\mathrm{N}} \sum_{j=1}^{\mathrm{N}} \delta_{\mathrm{ij}} \sum_{k=1}^{\mathrm{K}} \mathrm{M}_i(k) \mathrm{M}_j(k) / \mathrm{C}(k) \quad (2.5)$$

where $\mathrm{C}(k) = \sum_{i=1}^{\mathrm{N}} \mathrm{M}_i(k)$ is the expected number of points in the *k*-th cluster, and $\delta_{\mathrm{ij}}\mathrm{D}(\mathrm{i},\mathrm{j})$ is the pairwise distance (or dissimilarity) between points *i* and *j*. Equation (2.5) is minimized for the cluster probabilities $\mathrm{M}_i(k)$ so that point *i* belongs to cluster *k*. The deterministic annealing can be derived from an informatics theoretic (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998) or a physics formalism (Hofmann & Buhmann, 1997). In latter case one smoothes out the cost function (2) by integrating it with the Gibbs distribution exp(-H/T) over all degrees of freedom. This implies that one is minimizing not H but the free energy F at temperature T and entropy S.

$$\mathrm{F} = \mathrm{H} - \mathrm{TS} \quad (2.6)$$

As explained in detail in (Rose K. , Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems, 1998), the temperature T can be interpreted as a distance scale so that gradually reducing the temperature T in equations (2.5) and (2.6) corresponds to increasing resolution when one considers distance structure.

Our parallel implementations of equations (2.5) and (2.6) are quite mature, and have been used extensively (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008) (Fox, et al., 2009), although there is ongoing activity to improve the overall infrastructure to support the many linked runs needed. There are also some sophisticated options in these methods that are still being implemented. Figure 7 indicates that we perform manual hierarchical analyses already but, as explained in the next subsection, we propose to

build on current technology to dramatically improve the scaling to large numbers of sequences, which is currently limited by $O(N^2)$ complexity for all stages of the processing pipeline (Figure 1).

2.3 Metagenomics Studies with Hierarchical MDS

Even though the annealing algorithm is looking at genes with a decreasing distance scale, the current clustering algorithm is of $O(N^2)$ complexity, and its behavior (as distance scale is lowered) tracks the "worst region" because the criteria for success are not localized but rather summed over all regions. We can develop new algorithms and dissociate convergence criteria in different regions by a fully hierarchical algorithm using ideas from particle dynamics simulations O(NlogN). The essential idea is to run MDS on a subset of data (say 20,000 to 100,000 sequences) to map sequences to a 3D Euclidean space. This would take hours to days on our 768 core cluster. Then use orthogonal recursive bisection to divide 3D mapped space into geometrically compact regions. This allows both MDS mapping of the full dataset and a proper decomposition to allow efficient hierarchical clustering to find detailed sequence families. The best way to define regions is a research issue with both visual interface and automatic methods such as those described in the next paragraph possible.

We want to find a way of converting $O(N^2)$ algorithms like MDS to O(NlogN) in the same way that "Barnes-Hut trees" and "Fast Multipole" methods convert $O(N^2)$ to O(NlogN) in particle dynamics. The latter is described in many places (Barnes-Hut Simulation, 2009) including the PhD thesis of John Salmon (Salmon, 1991). The idea is illustrated in Figure 8, which shows a 2D projection of the data of Figure 7(a) with a hierarchical quad tree superimposed. This was generated by an open source Barnes Hut Tree code (Berg, 2009).

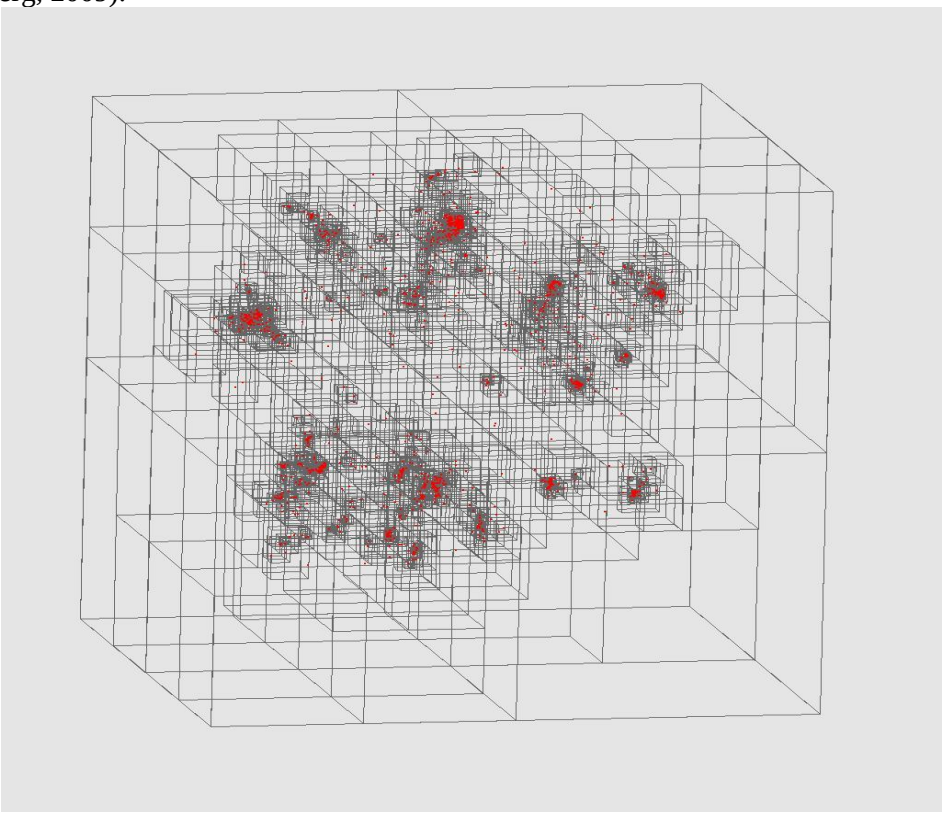


Figure 8. Barnes-Hut Oct tree generated from MDS dimension reduced Metagenomics data

This approach can be stated as follows: consider a collection of points, which are MDS mapped points in our case, but could be galaxies or stars in astrophysics. Divide them up by "orthogonal recursive bisection" to produce quad-trees in 2D or oct-trees in 3D (as in Figure 7). This is impractical in high dimensional space (as it generates 2^d children at each node in dimension d) but quite feasible in two or three dimensions. In our case, we need to use MDS to map the original sequences, which are both high dimensional and without defined universal vector representations (until we solve Multiple Sequence Alignment MSA problem). Note that the orthogonal recursive bisection divides space into regions where all points in a region are near each other. Given the nature of MDS, "near each other" in 3D implies "near each other" in original space. Observe that this approach builds a tree, and each internal or terminal node is a cubic region. User-defined criteria (such as number of points in region) establish if this region should be further split into 8 (in 3D) other regions. These criteria need to be designed for MDS, but are naturally the number of points as this determines the performance of part d) of algorithm below.

Our proposed algorithm proceeds as follows:

a) Perform a full $O(N_{subset}^2)$ MDS for a subset N_{subset} of points.
b) Find centers or representative (consensus) points for each region in oct-tree. The centers could be found in at least two ways. First one can use heuristic algorithms to find the center in the original space. A simple heuristic that has been successful for us is to find the point in the region with the minimum value for maximum distance from all other points in that region. A second approach is to find the geometric center in the mapped 3D MDS space (or a set of representative points around a 3D region), then find the nearest sequences to the center – these become representative points for the region. One does this for all nodes of the tree – not just the final nodes. Note that representative points are defined both in original space and in MDS mapped 3D space.
c) We now have a tree with regions and consensus points (centers) associated with all nodes. The next step is performed for remaining $N - N_{subset}$ points – call a typical point *p*. Take each of the points *p* and start at the top of the tree. Calculate distance of *p* to each tree node at a given level. Assign *p* to the node with the minimum distance from its center to *p* and continue down the tree looking at 8 sub-nodes below this node. Continue until you have reached bottom of tree.
d) Now we have assigned each $N - N_{subset}$ points *p* to a region of 3D space – let the assigned region have N_{region} points. Now one performs MDS within this region using $1+ N_{region}$ points (*p* plus points in region) to define the MDS mapping of point *p* with the mapping of original N_{region} points fixed. Note our criterion for dividing nodes probably includes $N_{region} <$ a cutoff value so we can control computational complexity of this step d). The criterion appropriate for dividing nodes is as mentioned above, an important research issue.

There are many refinements that can be evaluated. Maybe our point *p* is at boundary of a region. This could be addressed by either looking at quality of fit in step d) or the discrimination of distance test in step c). One would choose if necessary not a terminal node at step c) but the larger region corresponding to an internal node. There are tradeoffs between performance (which scales like the number of points in chosen region) and accuracy of method, which need to be researched. Finally all steps of the pipeline in Figure 1 have choices of heuristic parameters that can be significant and need study. In particular there is the optimal choice for weight functions and distance measures to use in equations (2.4) and (2.5).

Step a) of the above algorithm has time complexity of $O(N_{subset}^2)$. MDS interpolation to get the full dataset mapped to 3D (steps c) and d)) has a complexity that depends on implementation -- simplest is $O((N - N_{subset}) (\log N_{subset} + N_{region}))$. Clustering and MDS can use the same tree decomposition but clustering will likely use different size (larger) regions. The complexity for clustering is $O(N^2/\#\text{ regions})$ for the simplest algorithm. We call this approach "stage 1" and it will be our initial focus as it can reuse a lot of our existing software base.

Later we will attempt to derive a properly O(NlogN) clustering approach that is based, not on clustering sequences, but rather on clustering "regions" and sequences. Here we will modify the double sum over sequences in equation (2.5). The form of sum is preserved if sequences are nearby, but replaces sequence-sequence interaction by sequence-region or region-region terms if sequences are far away. Here, a region can be represented by its consensus (mean in particle dynamics) sequence for regions with a weight equal to number of sequences in region. The further sequences are apart, the larger regions can be (i.e. one chooses representation nearer the root of the tree formed by orthogonal recursive bisection). We call this speculative approach "stage 2". It is a natural generalization of the O(N) or O(NlogN) particle dynamics algorithms (Fox, Williams, & Messina, 1994) (Barnes-Hut Simulation, 2009).

This approach needs totally new software and significant algorithm work -- in particular to develop an error estimate to decide which level in tree (region size) to use -- this is done by multipole expansion in particle case. It can possibly provide not just scalable O(NlogN + N_{subset}^2) clustering but also a viable approach to Multiple Sequence Alignment for large numbers of sequences.

2.4 Summary

This section has discussed our parallel datamining algorithms for dimension reduction and clustering and their application to Metagenomics and PubChem chemical properties. We stressed the importance of methods that can be applied to cases where there is an underlying high dimension vector space and to cases where there are no defined vectors and only pairwise dissimilarities are known. We compared GTM and MDS showing they obtained similar representations for the PubChem case where vectors are known. The powerful non-vector algorithms scale like O(N^2) which makes them computationally expensive for large problems (with millions of points). We discussed some initial ideas for reducing this complexity to O(NlogN) which could be very important in enabling analysis of the large datasets expected from next generation of sequencers.

3 INNOVATIONS IN PROGRAMMING MODELS USING CLOUD TECHNOLOGIES

3.1 Runtimes and Programming Models

This section presents a brief introduction to a set of parallel runtimes we use in our evaluations. Specifically, we compare features of MapReduce programming models such as Hadoop and Dryad with traditional MPI and i-*MapReduce* as an extension of MapReduce model.

3.1.1 Parallel frameworks

3.1.1.1 Hadoop

Apache Hadoop (Apache Hadoop, 2009) has a similar architecture to Google's MapReduce runtime (Dean & Ghemawat, 2008), where it accesses data via HDFS, which maps all the local disks of the compute nodes to a single file system hierarchy, allowing the data to be dispersed across all the data/computing nodes. HDFS also replicates the data on multiple nodes so that failures of any nodes containing a portion of the data will not affect the computations which use that data. Hadoop schedules the MapReduce computation tasks depending on the data locality, improving the overall I/O bandwidth. The outputs of the *map* tasks are first stored in local disks until later, when the *reduce* tasks access them (pull) via HTTP connections. Although this approach simplifies the fault handling mechanism in Hadoop, it adds a significant communication overhead to the intermediate data transfers, especially for applications that produce small intermediate results frequently.

3.1.1.2 Dryad

Dryad (Isard, Budiu, Yu, Birrell, & Fetterly, 2007) is a distributed execution engine for coarse grain data parallel applications. Dryad considers computation tasks as directed acyclic graphs (DAG) where the vertices represent computation tasks and the edges act as communication channels over which the data flow from one vertex to another. In the HPC version of DryadLINQ the data is stored in (or partitioned to) Windows shared directories in local compute nodes and a meta-data file is use to produce a description of the data distribution and replication. Dryad schedules the execution of vertices depending on the data locality. (Note: The academic release of Dryad only exposes the DryadLINQ (Yu, et al., 2008) API for programmers. Therefore, all our implementations are written using DryadLINQ although it uses Dryad as the underlying runtime). Dryad also stores the output of vertices in local disks, and the other vertices which depend on these results, access them via the shared directories. This enables Dryad to re-execute failed vertices, a step which improves the fault tolerance in the programming model.

3.1.1.3 i-*MapReduce*

i-MapReduce (Ekanayake, Pallickara, & Fox, MapReduce for Data Intensive Scientific Analyses, 2008) (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008) is a light-weight MapReduce runtime (an early version was called CGL-MapReduce) that incorporates several improvements to the MapReduce programming model such as (i) faster intermediate data transfer via a pub/sub broker network; (ii) support for long running *map/reduce* tasks; and (iii) efficient support for iterative MapReduce computations. The use of streaming enables i-MapReduce to send the intermediate results directly from its producers to its consumers, and eliminates the overhead of the file based communication mechanisms adopted by both Hadoop and DryadLINQ. The support for long running *map/reduce* tasks enables configuring and re-using of *map/reduce* tasks in the case of iterative MapReduce computations, and eliminates the need for the re-configuring or the re-loading of static data in each iteration.

3.1.1.4 MPI Message Passing Interface

MPI (MPI, 2009), the de-facto standard for parallel programming, is a language-independent communications protocol that uses a message-passing paradigm to share the data and state among a set of cooperative processes running on a distributed memory system. The MPI specification defines a set of routines to support various parallel programming models such as point-to-point communication, collective communication, derived data types, and parallel I/O operations. Most MPI runtimes are deployed in computational clusters where a set of compute nodes are connected via a high-speed network connection yielding very low communication latencies (typically in microseconds). MPI processes typically have a direct mapping to the available processors in a compute cluster or to the processor cores in the case of multi-core systems. We use MPI as the baseline performance measure for the various algorithms that are used to evaluate the different parallel programming runtimes. Table 1 summarizes the different characteristics of Hadoop, Dryad, i-MapReduce, and MPI.

Table 1. Comparison of features supported by different parallel programming runtimes.

Feature	Hadoop	DryadLINQ	i-MapReduce	MPI
Programming Model	MapReduce	DAG based execution flows	MapReduce with a *Combine* phase	Variety of topologies constructed using the rich set of parallel constructs
Data Handling	HDFS	Shared directories/	Shared file system /	Shared file

		Local disks	Local disks	systems
Intermediate Data Communication	HDFS/ Point-to-point via HTTP	Files/TCP pipes/ Shared memory FIFO	Content Distribution Network (NaradaBrokering (Pallickara and Fox 2003))	Low latency communication channels
Scheduling	Data locality/ Rack aware	Data locality/ Network topology based run time graph optimizations	Data locality	Available processing capabilities
Failure Handling	Persistence via HDFS Re-execution of map and reduce tasks	Re-execution of vertices	Currently not implemented (Re-executing map tasks, redundant reduce tasks)	Program level Check pointing OpenMPI , FT MPI
Monitoring	Monitoring support of HDFS, Monitoring MapReduce computations	Monitoring support for execution graphs	Programming interface to monitor the progress of jobs	Minimal support for task level monitoring
Language Support	Implemented using Java. Other languages are supported via Hadoop Streaming	Programmable via C# DryadLINQ provides LINQ programming API for Dryad	Implemented using Java Other languages are supported via Java wrappers	C, C++, Fortran, Java, C#

3.1.2 Science in clouds – dynamic virtual clusters

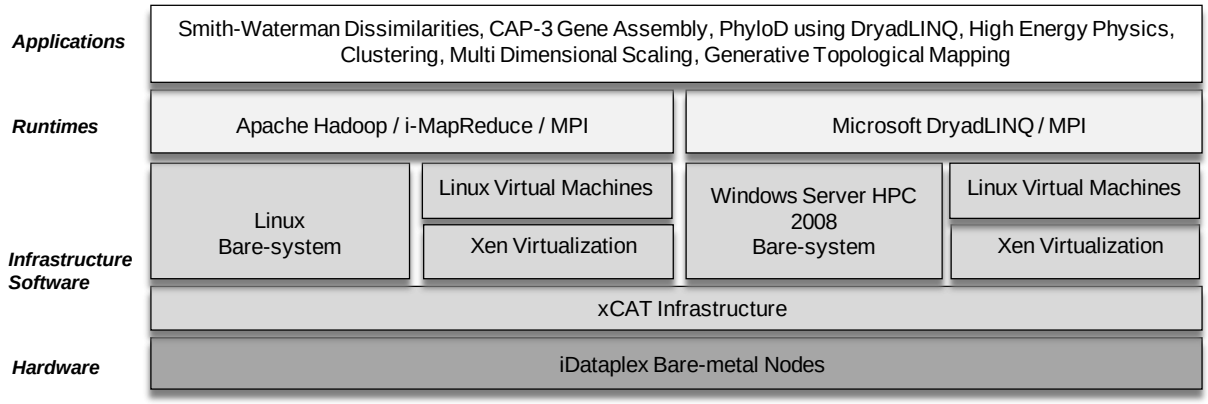


Figure 9. Software and hardware configuration of dynamic virtual cluster demonstration. Features include virtual cluster provisioning via xCAT and support of both stateful and stateless OS images.

Deploying virtual or bare-system clusters on demand is an emerging requirement in many HPC centers. The tools such as xCAT (xCAT, 2009) and MOAB (Moab Cluster Tools Suite, 2009) can be used to provide these capabilities on top of physical hardware infrastructures. In this section we discuss our experience in demonstrating the possibility of provisioning clusters with parallel runtimes and use them for scientific analyses.

We selected Hadoop and DryadLINQ to demonstrate the applicability of our idea. The SW-G application described in section 3.2 is implemented using both Hadoop and DryadLINQ. With bare-system and XEN (Barham, et al., 2003) virtualization, Hadoop running on Linux, and DryadLINQ running on Windows Server 2008 operating systems, we produced four operating system configurations; namely (i) Linux Bare System, (ii) Linux on XEN, (iii) Windows Bare System, and (iv) Windows on

XEN. Out of these four configurations, the fourth one did not work well due to the unavailability of the appropriate para-virtualization drivers. Therefore we selected the first three operating system configurations for this demonstration.

We selected xCAT infrastructure as our dynamic provisioning framework and set it up on top of bare hardware of a compute cluster. Figure 9 shows the various software/hardware components in our architecture. To implement the dynamic provisioning of clusters, we developed a software service that accept user inputs via a pub-sub messaging infrastructure and issue xCAT commands to switch a compute cluster to a given configuration. We installed Hadoop and DryadLINQ in the appropriate operation system configurations and developed initialization scripts to initialize the runtime with the start of the compute clusters. These developments enable us to provide a fully configured computation infrastructure deployed dynamically at the requests of the users.

We setup the initialization scripts to run SW-G pairwise distance calculation application after the initialization steps. This allows us to run a parallel application on the freshly deployed cluster automatically.

We developed a performance monitoring infrastructure to monitor the utilization (CPU, memory etc.) of the compute clusters using a pub-sub messaging infrastructure. The architecture of the monitoring infrastructure and the monitoring GUI are shown in Figure 10.

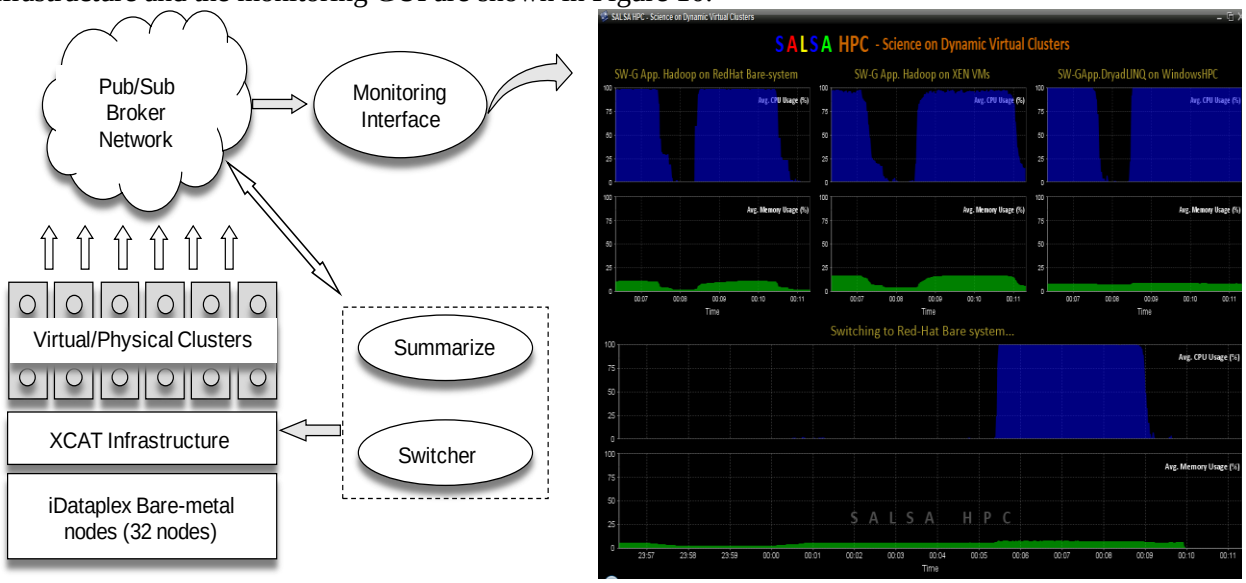


Figure 10. Architecture of the performance monitoring infrastructure and the monitoring GUI.

In the monitoring architecture, a daemon is placed in each computer node of the cluster, and is started with the initial boot sequence. All the monitor daemons send the monitored performances to a summarizer service via the pub-sub infrastructure. The summarizer service produces a global view of the performance of a given cluster and sends this information to a GUI that visualizes the results in real-time. The GUI is specifically developed to show the CPU and the memory utilization of the bare-system/virtual clusters when they are deployed dynamically.

With all the components in place, we implemented SW-G application running on dynamically deployed bare-system/virtual clusters with Hadoop and DryadLINQ parallel frameworks as shown in the performance chart of figure 10. We divide cluster nodes into three groups, each has 4 nodes of 16 core to run Hadoop on Redhat bare system, Hadoop on Xen VMs, and DryadLINQ on Windows HPC respectively. The average switching time between these three settings is about 5 minutes. (Ekanayake,

Gunarathne, & Qiu, Cloud Technologies for Bioinformatics Applications, 2010) observes a performance degradation between 15% and 25% for Hadoop SW-G application on Xen VMs. This dynamic virtual clusters environment will be extended in the FutureGrid project (FutureGrid Homepage, 2009).

3.2 Pairwise sequence alignment using Smith-Waterman-Gotoh

3.2.1 Introduction to Smith-Waterman-Gotoh (SWG)

Smith-Waterman (Smith & Waterman, 1981) is a widely used local sequence alignment algorithm for determining similar regions between two DNA or protein sequences. In our studies we use Smith-Waterman algorithm with Gotoh's (Gotoh, 1982) improvement for Alu sequencing. The Alu clustering problem (Price, Eskin, & Pevzner, 2004) is one of the most challenging problems for sequencing clustering because Alus represent the largest repeat families in human genome. As in metagenomics, this problem scales like $O(N^2)$ as given a set of sequences we need to compute the similarity between all possible pairs of sequences.

3.2.2 Implementations

3.2.2.1 Dryad Implementation

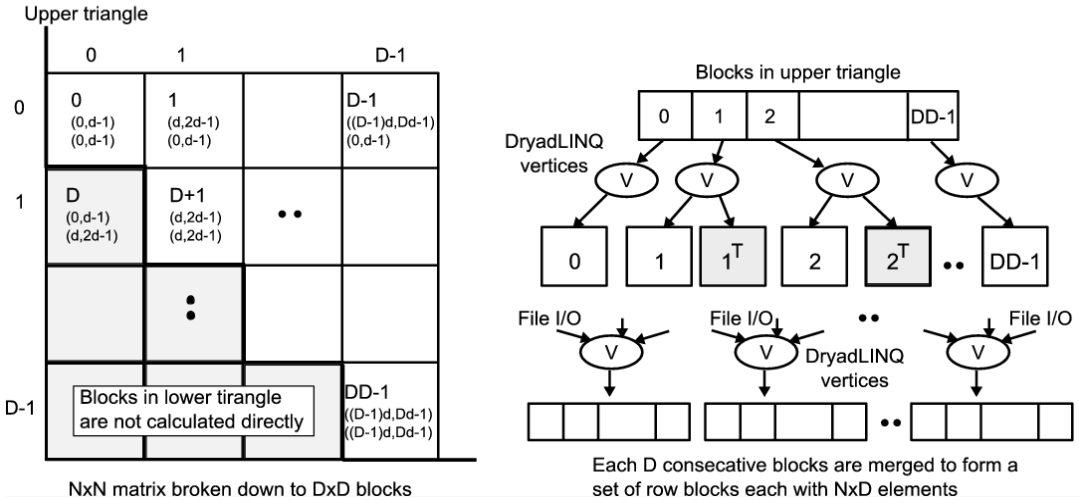


Figure 11. (a) Task decomposition (b) DryadLINQ implementation and vertex hierarchy

We developed a DryadLINQ application to perform the calculation of pairwise SW-G distances for a given set of genes by adopting a coarse grain task decomposition approach which requires minimum inter-process communication to ameliorate the higher communication and synchronization costs of the parallel runtime. To clarify our algorithm, let's consider an example where N gene sequences produce a pairwise distance matrix of size NxN. We decompose the computation task by considering the resultant matrix and group the overall computation into a block matrix of size DxD, where D is a multiple (>2) of the available computation nodes. Due to the symmetry of the distances δ_{ij} and= δ_{ji} we only calculate the distances in the blocks of the upper triangle of the block matrix as shown in Figure 11(a). The blocks in the upper triangle are partitioned (assigned) to the available compute nodes, and an "Dryad Apply" operation is used to execute a function to calculate (N/D)x(N/D) distances in each block. After computing the distances, the function calculates the transpose matrix of the result matrix which corresponds to a

block in the lower triangle, and writes both these matrices into two output files in the local file system. The names of these files and their block numbers are communicated back to the main program. The main program sorts the files based on their block numbers and then performs another "Apply" operation to combine the files corresponding to a row of blocks into a single large row block as shown in the Figure 11(b).

3.2.2.2 MPI Implementation

The MPI version of SW-G calculates pairwise distances using a set of either single or multi-threaded processes. We use the "Space Filling" MPI algorithm for data decomposition as shown in Figure 12(a), which runs a "space filling curve through lower triangular matrix" to produce equal numbers of pairs for each parallel unit (such as process or thread). For N gene sequences, we need to compute half of the values (in the lower triangular matrix), for a total of M = N x (N-1) /2 distances. At a high level,

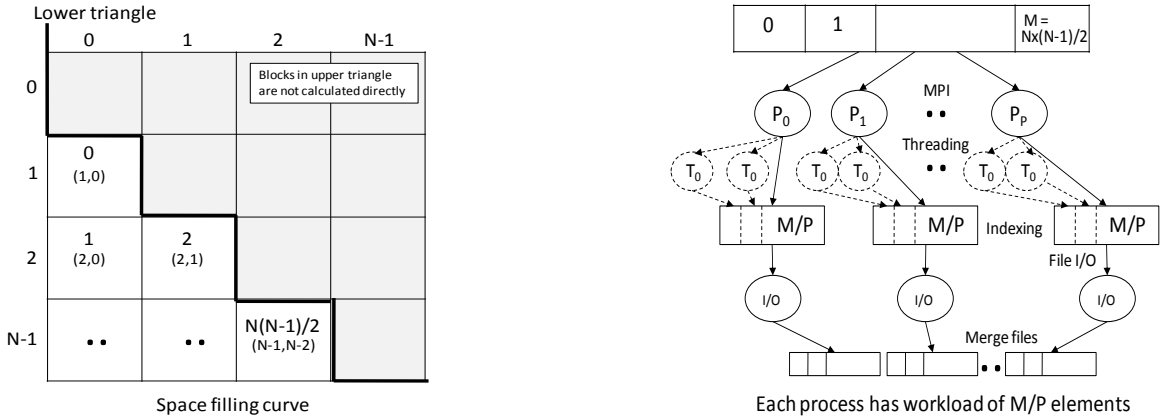


Figure 12. (a) Space Filling data decomposition *(b) MPI runtime processes architecture*

computation tasks are evenly divided among P processes and executed in parallel. The computation workload per process is M/P. At a low level, each computation task can be further divided into subgroups and run in T concurrent threads. Our implementation in Figure 12(b) is designed for flexible use of shared memory multicore system and distributed memory clusters (tight to medium tight coupled communication technologies such threading and MPI).

3.2.2.3 Apache Hadoop Implementation

We developed an Apache Hadoop version of the pairwise distance calculation program based on the JAligner (JAligner, 2009) program, the java implementation of the NAligner code used in Dryad version. Similar to the other implementations, the computation is partitioned in to blocks based on the resultant matrix. Each of the blocks would get computed as a map task. The block size (D) can be specified via an argument to the program. The block size needs to specified in such a way that there will be much more map tasks than the capacity of the system, so that the Apache Hadoop scheduling will happen as a pipeline of map tasks resulting in global load balancing of the application. The input data is distributed to the worker nodes through the Hadoop distributed cache, which makes them available in the local disk of each compute node.

A load balanced task partitioning strategy (as described below) is used to identify the blocks that need to be computed (dark grey) through map tasks as shown in the Figure 13(a). In addition all the blocks in the diagonal (light grey) are also computed. Even though the task partitioning mechanisms are different,

both Dryad-SWG and Hadoop-SWG ends up with essentially identical computation blocks, if the same block size is given to both the programs.

When $\beta >= \alpha$, *we calculate* $D(\alpha,\beta)$ *only if* $\alpha+\beta$ *is even,*
When $\beta < \alpha$, *we calculate* $D(\alpha,\beta)$ *only if* $\alpha+\beta$ *is odd.*

Figure 13(b) depicts the run time behavior of the Hadoop-swg program. In the given example the map task capacity of the system is "k" and the number of blocks is "N". The solid black lines represent the starting state, where "k" map tasks (blocks) will get scheduled in the compute nodes. The dark dashed lines represent the state at t_1, when two map tasks, m_2 & m_6, get completed and two map tasks from the pipeline gets scheduled for the placeholders emptied by the completed map tasks. The grey dotted lines represent the future.

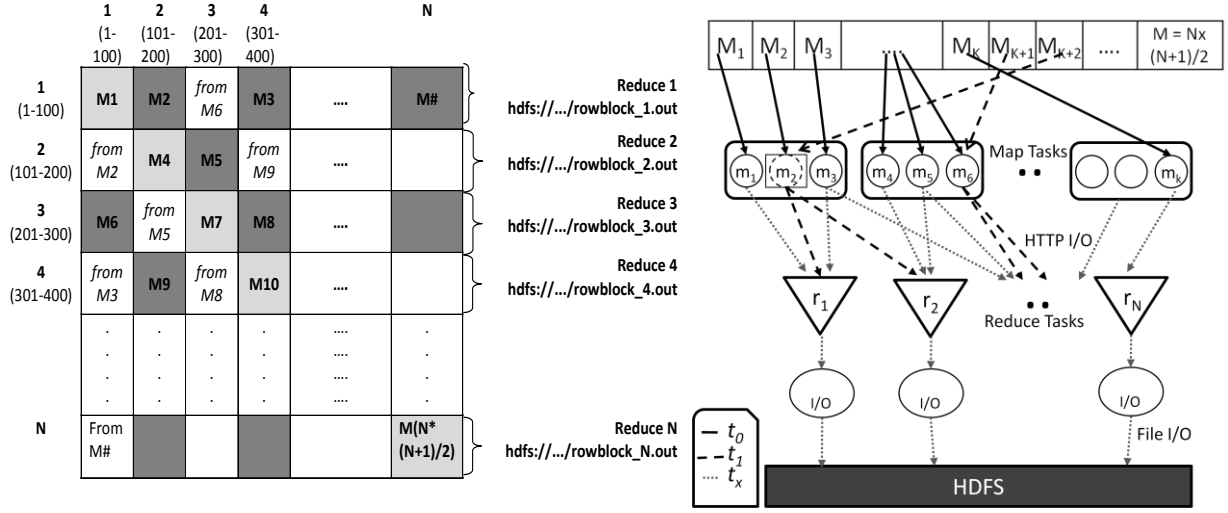


Figure 13. (a)Task decomposition *(b)Map and Reduce Tasks for Hadoop*

Map tasks use custom Hadoop writable objects as the output values to store the calculated pairwise distance matrices for the respective blocks. In addition, non-diagonal map tasks output the inverse distances matrix as a separate output value. Hadoop uses local files and http transfers to send the map task output key value pairs to the reduce tasks.

The outputs of the map tasks are then collected by the reduce tasks. Since the reduce tasks start collecting the outputs as soon as the first map task finishes, and continue to do so while other map tasks are executing, the data transfers from the map tasks to reduce tasks do not present a significant performance overhead to the program. The program currently creates a single reduce task per each row block resulting in total of (no. of sequences/block size) Reduce tasks. Each reduce task to accumulate the output distances for a row block and writes the collected output to a single file in Hadoop Distributed File System (HDFS). This results in N number of output files corresponding to each row block, similar to the output we produce in the Dryad version.

3.2.3 Performance comparison

We compared the Dryad, Hadoop and MPI implementations of ALU SW-G distance calculations using a replicated data set and obtained the following results. The data sets were generated by taking a 10000 sequence random sample from a real data set, and replicating it 2-5 times. Dryad and MPI tests were

performed in cluster *ref D* (Table 2), and the Hadoop tests were performed in cluster *ref A* (Table 2) Cluster ref A is identical to cluster *ref D*, which are two identical Windows HPC and Linux clusters. The Dryad & MPI results were scaled to account for the performance difference of the kernel programs, NAligner and the JAligner in their respective environments, for fair comparison with the Hadoop implementation.

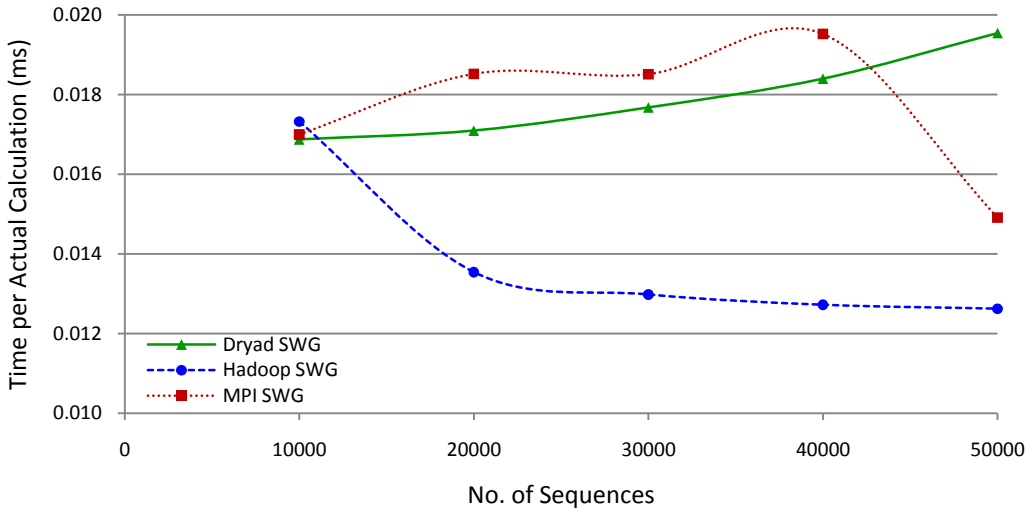


Figure 14. Comparison of Dryad, MPI and Hadoop technologies on ALU sequencing application with SW-G algorithm

Figure 14 indicates that all three implementations perform and scale well for this application with Hadoop implementation showing the best scaling. As expected, the times scaled proportionally to the square of the number of distances. On 256 cores the average time of 0.017 milliseconds per pair for 10k data set corresponds to roughly 4.5 milliseconds per pair calculated per core used. The coarse grained Hadoop & Dryad applications perform and scale competitively with the tightly synchronized MPI application.

We notice that the Hadoop implementation shows improved performance with the increase of the data set size, while Dryad performance degrades a bit. Hadoop improvements can be attributed to the diminishing of the framework overheads, while the Dryad degradation can be attributed to the memory management in the Windows and Dryad environment.

3.2.3.1 Inhomogeneous data study

Most of the data sets we encounter in the real world are inhomogeneous in nature, making it hard for the data analyzing programs to efficiently break down the problems. The same goes true for the gene sequence sets, where individual sequence lengths and the contents vary among each other. In this section we study the effect of inhomogeneous gene sequence lengths for the performance of our pairwise distance calculation applications.

$$SWG(A, B) = O(mn)$$

The time complexity to align and obtain distances for two genome sequences A, B with lengths *m* and *n* respectively using Smith-Waterman-Gotoh algorithm is approximately proportional to the product of the lengths of two sequences (O(*mn*)). All the above described distributed implementations of Smith-Waterman similarity calculation mechanisms rely on block decomposition to divide the larger problem space in to sub-problems that can be solved using the distributed components. Each block is assigned two

sub-sets of sequences, where Smith-Waterman pairwise distance similarity calculation needs to be performed for all the possible sequence pairs among the two sub sets. According to the previously mentioned time complexity of the Smith-Waterman kernel used by these distributed components, the execution time for a particular execution block depends on the lengths of the sequences assigned to the particular block.

Parallel execution frameworks like Dryad and Hadoop work optimally when the load is equally partitioned among the tasks. Depending on the scheduling strategy of the framework, blocks with different execution times can have an adverse effect on the performance of the applications, unless proper load balancing measures have been taken in the task partitioning steps. For an example, in Dryad, vertices are scheduled at the node level, making it possible for a node to have blocks with varying execution times. In this case if a single block inside a vertex takes a larger amount of time than other blocks to execute, then the whole node have to wait till the large task completes, which utilizes only a fraction of the node resources.

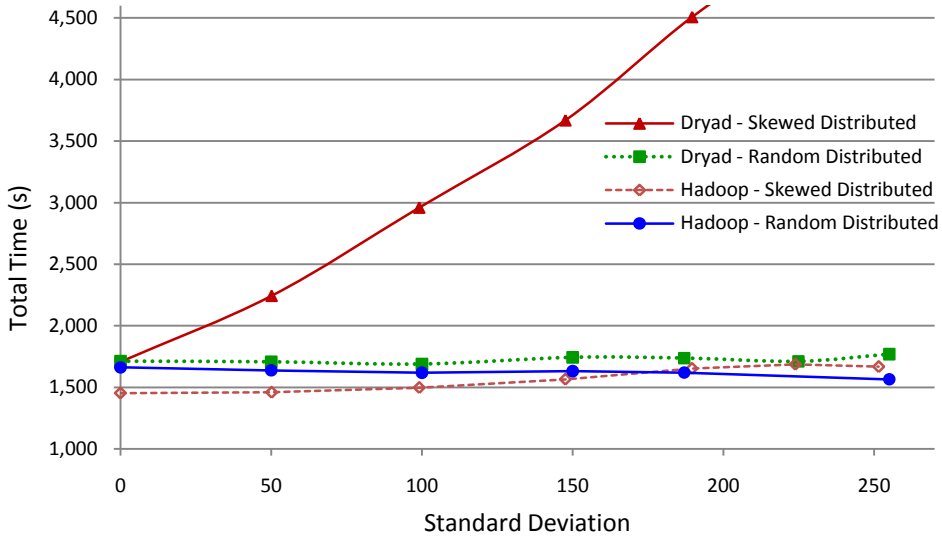


Figure 15. Performance of SW-G pairwise distance calculation application for randomly and skewed distibuted inhomogeneous data with '400' mean sequence length

Since the time taken for the Smith-Waterman pairwise distance calculation depends mainly on the lengths of the sequences and not on the actual contents of the sequences, we decided to use randomly generated gene sequence sets for this experiment. The gene sequence sets were obtained from a given mean sequence length (400) with varying standard deviations following a normal distribution of the sequence lengths. Each sequence set contained 10000 sequences leading to 100 million pairwise distance calculations to perform. We performed two studies using such inhomogeneous data sets. In the first study the sequences were randomly distributed in the data sets. In the second one the sequences were distributed using a skewed distribution, where the sequences in a set were arranged in the ascending order of their length.

Error! Reference source not found. presents the execution time taken for the randomly distributed and skewed distributed inhomogeneous data sets with the same mean length, by the two different implementations. The Dryad results depict the Dryad performance adjusted for the performance difference of the NAligner and JAligner kernel programs. As we notice from the **Error! Reference source not found.Error! Reference source not found.**, both Dryad and Hadoop performed satisfactorily for the randomly distributed inhomogeneous data, without showing significant performance

degradations with the increase of the standard deviation. This behavior can be attributed to the fact that the sequences with varying lengths are randomly distributed across a data set, effectively providing a natural load balancing to the execution times of the sequence blocks. In fact Hadoop implementation showed minor improvements in the execution times, which can be attributed to the fact that the actual workload gets reduced (effect of O(*mn*)) with the increase of the standard deviation even though the mean and the number of sequences stay the same.

For the skewed distributed inhomogeneous data, we notice a clear performance degradation in the Dryad implementation. Once again Hadoop performs consistently without showing significant performance degradation, even though it does not perform as well as its randomly distributed counterpart. This can be attributed to the global pipeline scheduling of the map tasks. In the Hadoop Smith-Waterman implementation, each block decomposition gets assigned to a single map task. The framework allows the administrator to specify the number of map tasks that can be run on a particular compute node. The global scheduler assigns the map tasks directly on to those placeholders in a much finer granularity than in Dryad, as and when the individual map tasks finish. This allows the Hadoop implementation to perform natural global level load balancing. In this case it might even be advantageous to have varying task execution times to iron out the effect of any trailing map tasks towards the end of the computation. Dryad implementation pre-allocates all the tasks to the compute nodes and does not perform any dynamic scheduling across the nodes. This makes a node which gets a larger work chunk to take considerable longer time than a node which gets a smaller work chunk, making the node with a smaller work chuck to idle while the other nodes finish.

3.3 Sequence assembly using Cap3

3.3.1 Introduction to Cap3

Cap3 (Huang & Madan, 1999) is a sequence assembly program which assembles DNA sequences by aligning and merging sequence fragments. The Cap3 algorithm works in several steps after reading a collection of gene sequences from an input file in the FASTA format. In the first two steps the poor regions of the fragments are removed, and the overlaps between the fragments are calculated. The third step identifies and removes the false overlaps. In the next step, the fragments are then joined to form what are called contigs or one or more overlapping DNA segments derived from a single genetic source. Finally, the last step constructs multiple sequence alignments and generates consensus sequences. This program outputs several files as well as standard output.

3.3.2 Implementations

Cap3 is often used with lots of input files making it an embarrassingly parallel application requiring no inter-process communications. We implemented parallel applications for Cap3 using Microsoft DryadLINQ (Yu, et al., 2008) (Isard, Budiu, Yu, Birrell, & Fetterly, 2007) and Apache Hadoop (Apache Hadoop, 2009). This fits as a “map only” application for the MapReduce model. The Hadoop application is implemented by creating map tasks which execute the Cap3 program as a separate process on the given input FASTA file. Since the Cap3 application is implemented in C, we do not have the luxury of using the Hadoop file system (HDFS) directly. Hence the data needs to be stored in a shared file system across the nodes. Therefore, we are actively investigating the possibility of using Hadoop streaming and mountable HDFS for this purpose.

For the DryadLINQ application, the set of input files are equally partitioned across the compute nodes, and stored in their local disks. A data partition file is created for each node containing the list of data files that resides in that particular node. We used the DryadLINQ “Select” operation to apply a function on the input files. The function will execute the Cap3 program passing the input file name together with other

parameters and will save the standard output from the program. All the outputs will get moved to a predefined location (an approach supported by both implementations).

3.3.3 Performance

First we performed a scalability test on our Cap3 implementations using a homogeneous data set. This data set is created by replicating a single file for a given number of times. The file we chose contained 458 sequences. When interpreting the results for Hadoop Cap3 implementation on Linux and DryadLINQ Cap3 implementation on windows, it should be noted that the Cap3 standalone program performed on average approximately 12.5% faster on windows than on Linux in the testing environment.

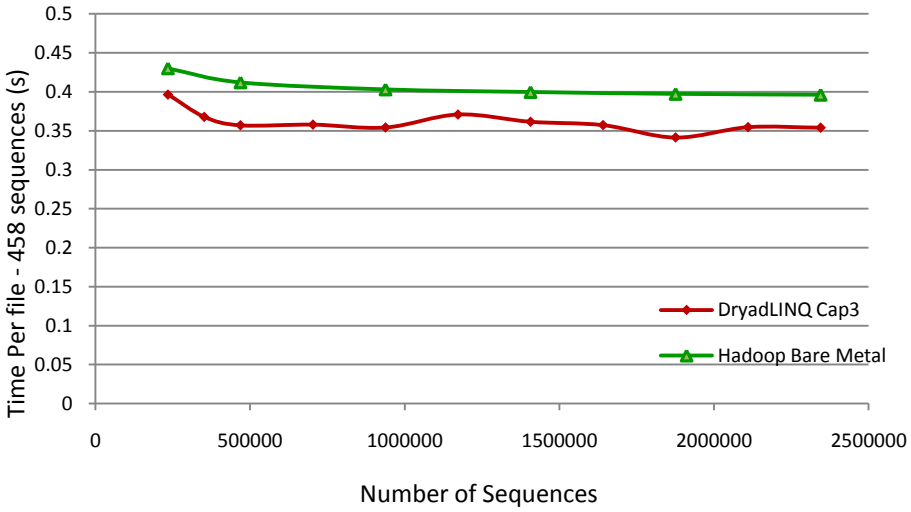


Figure 16. Cap3 scalability test with homogeneous data

As we can see from Figure 16, both Hadoop and DryadLINQ show good scaling for the Cap3 application, with even slightly increased performance with the increase of data size. The increase must be happening due to the overheads of the framework getting diminished over the larger workload. On 256 cores the average time 0.4 seconds on Hadoop to execute Cap3 program on a single data set corresponds to approximately 102 seconds per file executed per core, while the average time 0.36 seconds for DryadLINQ Cap3 application corresponds to approximately 92 seconds per file executed per core.

3.3.3.1 Inhomogeneous Data Study

Unlike the Smith-Waterman Gotoh case, Cap3 program execution time does not directly depend on the file size or the size of the sequences. It depends mainly on the content of the sequences. This makes it hard for us to artificially generate inhomogeneous data sets for the Cap3 program, therefore forcing us to use real data. When generating the data sets, first we calculated the standalone Cap3 execution time for each of the files in our data set. Then, based on those timings, we created data sets that have approximately similar mean times, while the standard deviation of the standalone running times is different in each data set. We performed the performance testing for randomly distributed as well as skewed distributed (sorted according to individual file running time) data sets similar to the SWG inhomogeneous study. The speedup is taken by dividing the sum of sequential running times of the files in the data set by the parallel implementation running time.

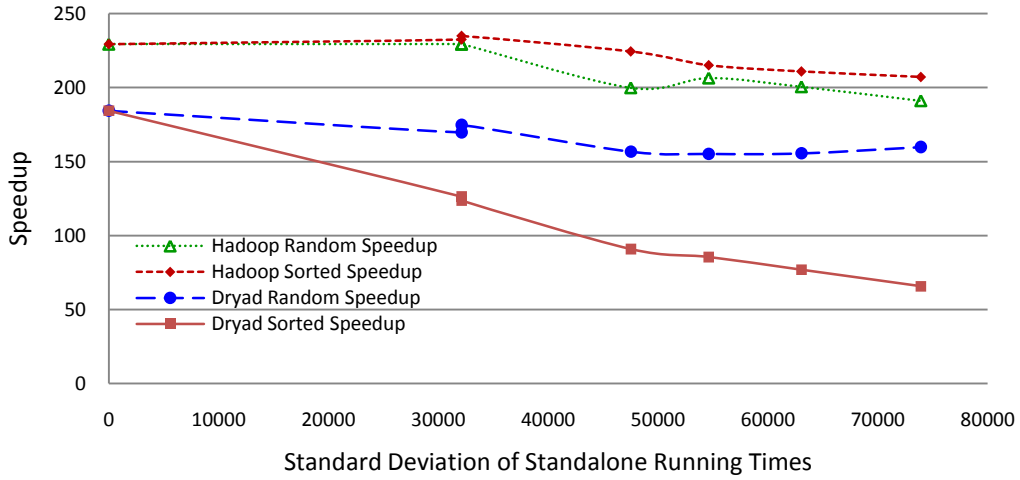


Figure 17. Cap3 inhomogeneous data performance

Figure 17 depicts the Cap3 inhomogeneous performance results for Hadoop & Dryad. Hadoop shows satisfactory scaling for both random as well as sorted data sets, while the Dryad shows satisfactory scaling in the randomly distributed data set. Once again we notice that Dryad does not perform well for the skewed distributed inhomogeneous data due to its' static non-global scheduling.

3.4 Summary

The heart of this section is the discussion of applying MapReduce to two problems – calculation of Smith-Waterman dissimilarities and CAP3 EST assembly – where current technologies can be used efficiently. We compare Hadoop (on Linux) to Dryad and MPI (on Windows). For homogeneous problems we find similar performance for all technologies but we find Hadoop currently has advantages over Dryad for inhomogeneous data. We also describe some useful technologies to allow rapid deployment of different operating environments on a given cluster hardware. This allows these tests to be executed fairly with all environments running on the same hardware.

4 ITERATIVE MAPREDUCE WITH I-*MAPREDUCE*

MapReduce is a programming model introduced by Google to support large scale data processing applications (Ghemawat, January, 2008). The simplicity of this programming model and the ease of supporting quality of services make it more suitable for large scale data processing applications. Our experience in applying MapReduce for scientific analyses reveals that it is suitable for many scientific analyses as well. However, we noticed that the current MapReduce programming model and its implementations such as Apache Hadoop do not support iterative MapReduce computations efficiently. Iterative computations are common in many fields such as data clustering, machine learning, and computer vision, and many of these applications can be implemented as MapReduce computations. In i-*MapReduce* (an early version was known as CGL-MapReduce) (Ekanayake, Pallickara, & Fox, MapReduce for Data Intensive Scientific Analyses, 2008) (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008) (i-MapReduce, 2009), we present an extended MapReduce programming model and a prototype implementation to support iterative MapReduce computations efficiently. Note (Chu, 2006) emphasized that the MapReduce approach is applicable to many data mining applications, but the performance will often be poor without the extra capabilities of i-*MapReduce*.

4.1 MapReduce Extensions

Many iterative applications we analyzed show a common characteristic of operating on two types of data products called static and variable data. Static data is used in each iteration and remain fixed throughout the computation, whereas the variable data is the computed results in each iteration and typically consumed in the next iteration step in many expectation maximization (EM) type algorithms. For example, if we consider K-means clustering algorithm (MacQueen), during the n^{th} iteration the program uses the input data set and the cluster centers computed during the $(n-1)^{th}$ iteration to calculate the next set of cluster centers.

Although some of the typical MapReduce computations such as distributed sorting, information retrieval and word histogramming consume very large data sets, many iterative applications we encounter operate on moderately sized data sets which can fit into the distributed memory of the computation clusters. This observation leads us to explore the idea of using long running map/reduce tasks similar to the long running parallel processes in many MPI applications which last throughout the life of the computation. The long running (cacheable) map/reduce tasks allow map/reduce tasks to be configured with static data and use them without loading them again and again in each iteration. Current MapReduce implementations such as Hadoop (Apache Hadoop, 2009) and DryadLINQ (Yu, et al., 2008) do not support this behavior. Hence they initiate new map/reduce tasks and load static data in each iteration step, therefore incurring considerable performance overheads. This distinction is shown in Figure 18. By supporting long running map/reduce tasks, we do not encourage users to store state information in the map/reduce tasks, which violates the "side-effect-free" nature of the map/reduce computations rather achieving considerable performance gains by caching the static data across map/reduce tasks. The framework does not guarantee the use of same set of map/reduce tasks throughout the life of the iterative computation.

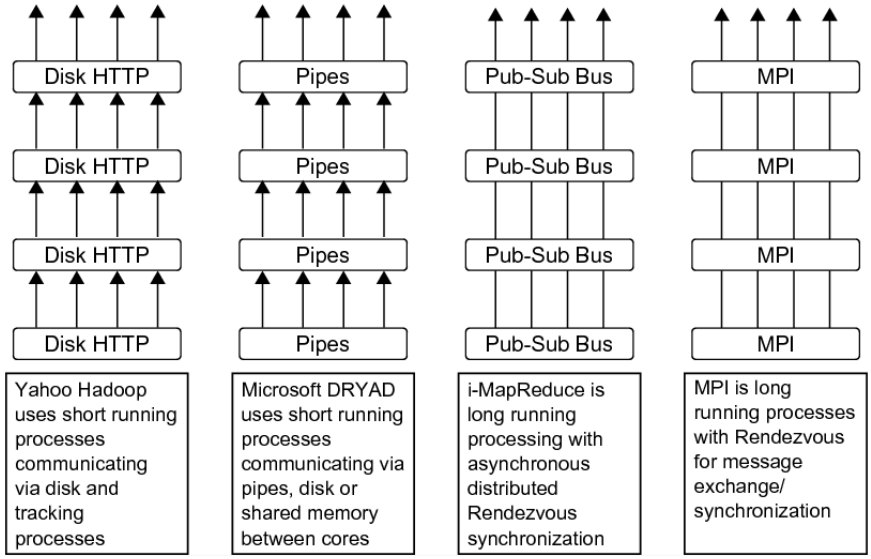


Figure 18. Long running and short running processes in various parallel programming runtimes.

In addition, we also add an optional reduction phase named "combine" to the MapReduce computation to allow programs to access the outputs of the reduce phase as a single value. Combine phase is another reduction phase which can be used to aggregate the results of the reduce phase into a single value. The user program and the combine operation run on a single process space allowing its output to be

directly accessible to the user program. This enables the user to check conditions based on the output of the MapReduce computations.

i-*MapReduce* uses streaming for all the communication/data transfer requirements, which eliminates the overhead in transferring data via file systems as in Hadoop or DryadLINQ. The output <Key,Value> pairs produced during the map stage get transferred directly to the reduce stage, and the output of the reduce stage get transferred directly to the combined stage via the pub-sub broker network. Currently i-*MapReduce* use the publish-subscribe messaging capabilities of NaradaBrokering (Pallickara & Fox, 2003) messaging infrastructure, but the framework is extensible to support any other publish-subscribe messaging infrastructure such as Active MQ (ActiveMQ, 2009).

We provide two mechanisms to access data in i-*MapReduce;* (i) from the local disk of the computer nodes, and (ii) directly from the pub-sub infrastructure. For the simplicity of the implementation, we provide a file based data access mechanism for the map/reduce tasks. The data distribution is left for the users to manage, and we plan to provide tools to perform such operations in the near future. Once distributed, i-*MapReduce* generates a meta-data file that can be used by the framework to run MapReduce computations. Apart from the above, the use of streaming enables i-*MapReduce* to support features such as directly sending input <Key,Value> pairs for the map stage, and configuring map/reduce stages using the data sent from the user program. Figure 19 shows the programming model of i-*MapReduce*, and how iterative MapReduce computations are executed.

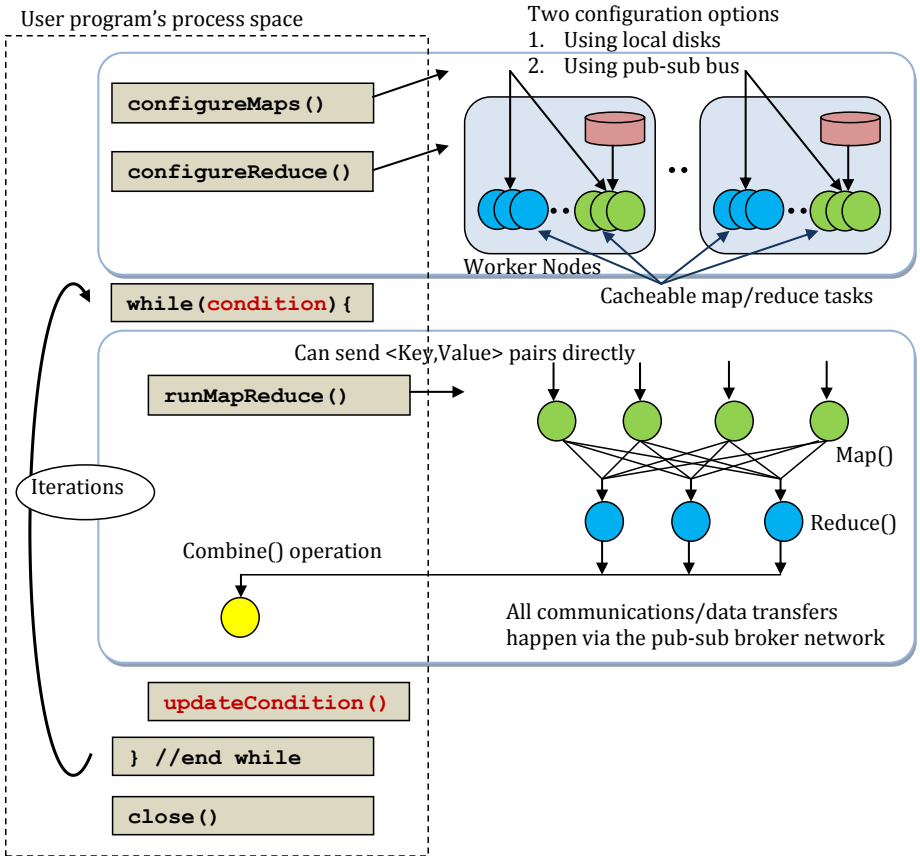


*Figure 19. Iterative MapReduce programming model using i-*MapReduce.

4.2 Performance of i-*MapReduce* for Iterative Computations

We have used the i-*MapReduce* framework to implement a series of scientific data analyses applications ranging from simple Map-only type operations to multiple iterative computations. Here we are presenting the results of four such applications, namely (i) CAP3 (Huang & Madan, 1999) gene sequence assembly, (ii) High Energy Physics data analysis, (iii) K-means clustering, and (iv) Matrix multiplication. We have also implemented the above applications using Apache Hadoop and DryadLINQ and MPI as well. The details of these applications and the parallel implementations are explained in more details in our previous publications (Fox, Bae, Ekanayake, Qiu, & Yuan, 2008). Figures 20 through Figure 23 present the results of our evaluations. Notice that, to obtain these results, we have used two compute clusters as shown in table 2. All the DryadLINQ applications were run on cluster ref. B while Hadoop, i-MapReduce and MPI applications were run on cluster ref. A. The overhead calculation is based on the formula (4.1) presented below.

$$\text{Overhead } f(p) = [p * T(p) - T(1)] / T(1) \qquad (4.1)$$

where p denotes the number of parallel processes used and T(p) denotes the time when p processes were used. T(1) gives the sequential time for the program.

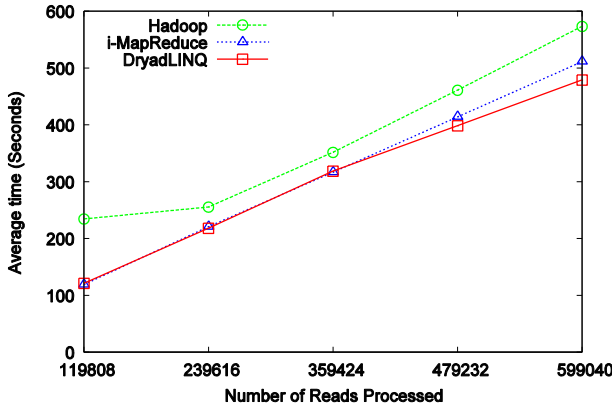


Figure 20. Performance of CAP3 gene assembly programs under varying input sizes.

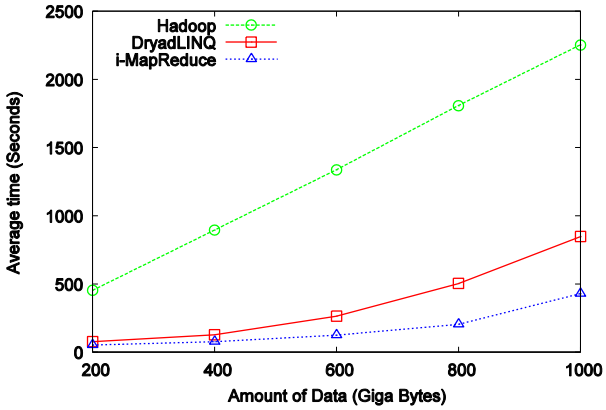


Figure 21. Performance of High Energy Physics programs under varying input sizes.

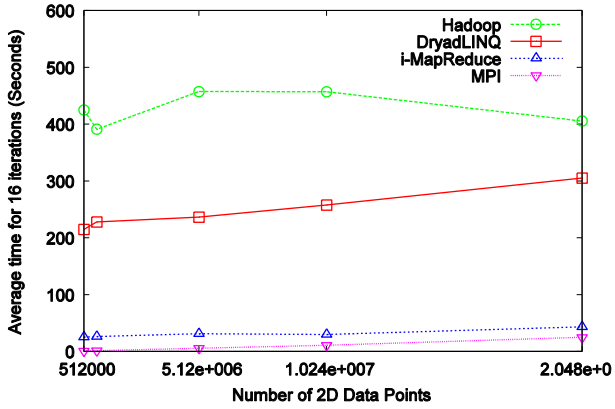


Figure 22. Overhead of K-means clustering implementations under

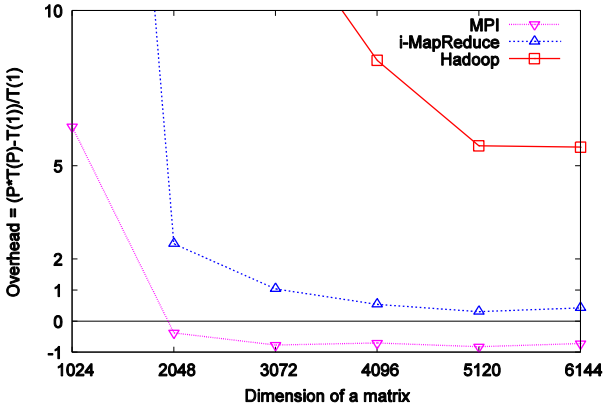


Figure 23. Overhead of matrix multiplication implementations under

varying input sizes. *varying input sizes.*

The CAP3 application case is a map-only (or typically named as pleasingly parallel) application in which the parallel processes require no inter process communications. The High Energy Physics (HEP) application is a typical MapReduce application aiming to produce a histogram of identified features from a large volume of data obtained during fusion experiments. The previous results indicate that all the runtimes, Hadoop, DryadLINQ and i-*MapReduce* perform equally well for these two types of applications. Notice that the higher running time observed in Hadoop in the case of HEP data analysis was due to the placement of data in a different parallel file system Hadoop's built in distributed file system HDFS (Apache Hadoop, 2009). This is because the ROOT (ROOT, Data Analysis Framework, 2009) data analysis framework used for HEP analysis could only read input files from local disks. Apart from above, these two analysis show that the i-*MapReduce* has not introduced any additional overheads for the typical MapReduce applications.

K-means clustering and matrix multiplication applications present typical iterative application characteristics. The graphs in Figure 22 and Figure 23 highlight the applicability of i-*MapReduce* to these iterative applications. The performance of i-*MapReduce* in the case of K-means clustering and the parallel overhead in the case of matrix multiplication are close to the values of MPI where both Hadoop and DryadLINQ shows relatively higher parallel overheads. Our approach of using long running map/reduce tasks and the use of streaming for the data transfers have eliminated many overheads present in other runtimes and enabled i-*MapReduce* to perform iterative MapReduce applications efficiently.

4.3 Related Work

MapReduce was first introduced in the Lisp programming language in which the developer is allowed to use a function to map a data set into another data set, and then use a function to reduce (combine) the results (G. L. Steel, 1995). J. Dean and S. Ghemawat introduced Google MapReduce and the associated programming model for large scale data intensive applications. Their framework supports fault tolerance, and is able to run on a large clusters built using commodity hardware. Swazall is an interpreted programming language for developing MapReduce programs based on Google's MapReduce implementation. R. Pike et al. present its semantics and its usability in their paper (Pike, Dorward, Griesemer, & Quinlan, 2005). The language is geared towards processing large document collections, which are typical operations for Google.

Sector/Sphere (Gu, 2009) is a parallel runtime developed by Y. Gu, and R. L. Grossman that can be used to implement MapReduce style applications. Sphere adopts a streaming based computation model used in GPUs which can be used to develop applications with parallel topologies as a collection of MapReduce style applications. Sphere stores intermediate data on files, and hence is susceptible to higher overheads for iterative applications.

Disco (Disco project, 2009) is an open source MapReduce runtime developed using a functional programming language named Erlang (Erlang programming language, 2009). Disco architecture shares clear similarities to the Google and Hadoop MapReduce architectures where it stores the intermediate results in local files and access them later using HTTP from the appropriate reduce tasks. However, Disco does not support a distributed file system as HDFS but expects the files to be distributed initially over the multiple disks of the cluster.

All the above runtimes focus on computations that can fit into a single cycle of MapReduce programming model. In i-*MapReduce* our focus is on iterative map reduce computations. Hence we introduce optimizations to the programming model and to the implementation to support these computations efficiently.

All-Pairs (Moretti, Bui, Hollingsworth, Rich, Flynn, & Thain, 2009) is an abstraction that can be used to solve a common problem of comparing all the elements in a data set with all the elements in another data set by applying a given function. This problem can be implemented using typical MapReduce frameworks such as Hadoop. We have shown a similar application in section 3.2.

M. Isard et al. present Dryad - a distributed execution engine for coarse grain data parallel applications (Isard, Budiu, Yu, Birrell, & Fetterly, 2007). It combines the MapReduce programming style with dataflow graphs to solve the computation tasks. DryadLINQ exposes a LINQ (LINQ Language-Integrated Query, 2009) based programming API for Dryad. The Directed Acyclic Graph (DAG) based programming model of Dryad can support more classes of applications than pure MapReduce programming model. DryadLINQ also provides a "loop unrolling" feature that can be used to create aggregated execution graphs combing a few iterations of iterative computations. However, as we have shown in Figure 22 it could not reduce the overhead of the programming model for large (in number of iterations) iterative applications.

4.4 Future Work on i-*MapReduce*

In our current research we are focusing on adding fault tolerance support for the i-*MapReduce* runtime since this is a key feature of Hadoop and Dryad. Saving system state at every iteration will add considerable overheads for iterative applications. Therefore we are working to add features to i-*MapReduce* so that it can save system state after a given number of iterations (Gropp & Lusk, 2004) (Fagg & Dongarra, 2000) (Hursey, Mattox, & Lumsdaine, 2009). Apart from the above we are researching further MapReduce extensions which will expand its use into more classes of parallel applications. We intend to support all applications that can be implemented using MPI Reduce, Broadcast and Synchronization primitives. We will present our findings under the umbrella project i-*MapReduce* (i-MapReduce, 2009).

4.5 Summary

In this section we have discussed our experience in developing an extended MapReduce programming model and a prototype implementation named i-*MapReduce*. We have shown that with i-*MapReduce* one can apply MapReduce to iterative applications and obtain considerable performance gains comparable to MPI implementations of the same applications.

5 CONCLUSION

We have discussed the use of innovative data-mining algorithms and new programming models for several Life Sciences applications. We are particularly interested in methods that are applicable to large data sets coming from high throughput devices of steadily increasing power. We have shown impressive results for both clustering and dimension reduction algorithms, and the use of MapReduce on modest size problems. However, we have identified two key areas where further research is essential. Therefore, we propose to develop new O(NlogN) complexity algorithms suitable for the analysis of millions of sequences. Furthermore, we suggest Iterative MapReduce as a promising programming model combining the best features of MapReduce with those of high performance environments such as MPI.

ACKNOWLEDGEMENTS

We would like to thank Microsoft for their collaboration and support. Tony Hey, Roger Barga, Dennis Gannon and Christophe Poulain played key roles in providing technical support. We appreciate our collaborators from IU School of Informatics and Computing. David Wild and Qian Zhu gave us important

feedback on visualization for PubChem data. We are grateful for UITS system administrators Joe Rinkovsky and Jenett Tillotson who set up the infrastructure for the SALSA virtual cluster environment.

APPENDIX A: DIFFERENT CLUSTER CONFIGURATIONS USED IN THIS ANALYSIS

Table 2. Different computation clusters used for this analysis.

Feature	Linux Cluster (Ref A)	Windows Cluster (Ref B)	Windows Cluster (Ref C)	Windows Cluster (Ref D)	Windows Cluster (Ref E)	Linux Cluster (Ref F)
CPU	Intel(R) Xeon(R) L5420 2.50GHz	Intel(R) Xeon(R) L5420 2.50GHz	Intel(R) Xeon(R) E7450 2.40GHz	Intel(R) Xeon(R) L5420 2.50GHz	AMD Opteron 8356 2.3 GHz	Intel(R) Xeon(R) E5345 2.33 GHz
# CPU # Cores	2 8	2 8	4 6	2 8	4 16	2 4
Memory	32 GB	16 GB	48 GB	32 GB	16 GB	20 GB
# Disk	1	2	1	1	1	1
Network	Giga bit Ethernet	Giga bit Ethernet	20 Gbps Infiniband or 1 Gbps	Giga bit Ethernet	Giga bit Ethernet	Giga bit Ethernet
Operating System	Red Hat Enterprise Linux Server release 5.3 -64 bit	Microsoft Window HPC Server 2008 (Service Pack 1) - 64 bit	Microsoft Window HPC Server 2008 (Service Pack 1) - 64 bit	Microsoft Window HPC Server 2008 (Service Pack 1) - 64 bit	Microsoft Window HPC Server 2008 (Service Pack 1) - 64 bit	GNU/Linux x86_64
# Cores	256	256	768	256	128	64

REFERENCES

ActiveMQ. (2009). Retrieved December 2009, from http://activemq.apache.org/

Apache Hadoop. (2009). Retrieved December 2009, from http://hadoop.apache.org/

Bae, S.-H. (2008). Parallel Multidimensional Scaling Performance on Multicore Systems. *Proceedings of the Advances in High-Performance E-Science Middleware and Applications workshop (AHEMA) of Fourth IEEE International Conference on eScience* (pp. 695-702). Indianapolis: IEEE Computer Society.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al. (2003). Xen and the art of virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing* (pp. 164-177). NY, USA: ACM Press.

Barnes-Hut Simulation. (2009). Retrieved December 2009, from http://en.wikipedia.org/wiki/Barnes-Hut_simulation

Berg, I. (2009). *Simulation of N-body problems with the Barnes-Hut algorithm.* Retrieved December 2009, from http://www.beltoforion.de/barnes_hut/barnes_hut_de.html

Bishop, C. M., & Svensén, M. (1997). GTM: A principled alternative to the self-organizing map. *Advances in neural information processing systems* , 354--360.

Bishop, C. M., Svensén, M., & Williams, C. K. (1998). GTM: The generative topographic mapping. *Neural computation , 10*, 215--234.

Borg, I., & Groenen, P. J. (2005). *Modern Multidimensional Scaling: Theory and Applications.* Springer.

Campbell, N., & Atchley, W. R. (1981). The geometry of canonical variate analysis. *Systematic Zoology* , 268--280.

Chu, C. T. (2006). Map-Reduce for Machine Learning on Multicore. *NIPS* (pp. 281--288). MIT Press.

de Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. *Recent Developments in Statistics* , 133-145.

de Leeuw, J. (1988). Convergence of the majorization method for multidimensional scaling. *Journal of Classification , 5*, 163-180.

Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Commun. ACM , 51* (1), 107-113.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum Likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B* , 1--38.

Disco project. (2009). Retrieved December 2009, from http://discoproject.org/

Ekanayake, J., Balkir, A., Gunarathne, T., Fox, G., Poulain, C., Araujo, N., et al. (2009). DryadLINQ for Scientific Analyses. *Fifth IEEE International Conference on eScience: 2009.* Oxford: IEEE.

Ekanayake, J., Gunarathne, T., & Qiu, J. (2010). Cloud Technologies for Bioinformatics Applications. *Invited paper submitted to the Journal of IEEE Transactions on Parallel and Distributed Systems* .

Ekanayake, J., Gunarathne, T., Qiu, J., Fox, G., Beason, S., Choi, J. Y., et al. (2009). *Applicability of DryadLINQ to Scientific Applications.* Community Grids Laboratory, Indiana University.

Ekanayake, J., Pallickara, S., & Fox, G. (2008). MapReduce for Data Intensive Scientific Analyses. *Fourth IEEE International Conference on eScience* (pp. 277-284). IEEE Press.

Ekanayake, J., Qiu, X., Gunarathne, T., Beason, S., & Fox, G. (2010). High Performance Parallel Computing with Clouds and Cloud Technologies. In *Cloud Computing and Software Services: Theory and Techniques.* CRC Press (Taylor and Francis), ISBN-10: 1439803153.

Erlang programming language. (2009). Retrieved December 2009, from http://www.erlang.org/

Fagg, G. E., & Dongarra, J. J. (2000). FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. *Lecture Notes in Computer Science 1908* (pp. 346-353). Springer Verlag.

Fox, G. C., Williams, R. D., & Messina, P. C. (1994). *Parallel computing works! (section http://www.old-npac.org/copywrite/pcw/node278.html#SECTION00144000000000000000).* Morgan Kaufmann Publishers, Inc.

Fox, G., Bae, S.-H., Ekanayake, J., Qiu, X., & Yuan, H. (2008). Parallel Data Mining from Multicore to Cloudy Grids. *High Performance Computing and Grids workshop.*

Fox, G., Qiu, X., Beason, S., Choi, J. Y., Rho, M., Tang, H., et al. (2009). Biomedical Case Studies in Data Intensive Computing. *The 1st International Conference on Cloud Computing (CloudCom 2009).* Springer Verlag.

FutureGrid Homepage. (2009). Retrieved December 2009, from http://www.futuregrid.org

G. L. Steel, J. (1995). Parallelism in Lisp. *SIGPLAN Lisp Pointers vol. VIII(2)* , 1-14.

Ghemawat, J. D. (January, 2008). Mapreduce: Simplified data processing on large clusters. *ACM Commun. vol 51* , 107-113.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology , 162*, 705-708.

Gropp, W., & Lusk, E. (2004). Fault Tolerance in Message Passing Interface Programs. *International Journal of High Performance Computing Applications , 18*, 363-372.

Gu, Y. G. (2009). Sector and Sphere: The Design and Implementation of a High Performance Data Cloud. *Crossing boundaries: computational science, e-Science and global e-Infrastructure I. Selected papers from the UK e-Science All Hands Meeting 2008 Phil. Trans. R. Soc. A , 367*, 2429-2445.

Hadoop Distributed File System HDFS. (2009). Retrieved December 2009, from http://hadoop.apache.org/hdfs/

Hardoon, D. R., Szedmak, S., & Shawe-Taylor, J. (2004). Canonical correlation analysis: an overview with application to learning methods. *Neural Computation , 16*, 2639--2664.

Hofmann, T., & Buhmann, J. M. (1997). Pairwise data clustering by deterministic annealing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on , 19*, 1--14.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika , 28*, 321--377.

Huang, X., & Madan, A. (1999). CAP3: A DNA sequence assembly program. *Genome Res. 9(9)* , 868-77.

Hursey, J., Mattox, T. I., & Lumsdaine, A. (2009). Interconnect agnostic checkpoint/restart in Open MPI. *Proceedings of the 18th ACM international symposium on High Performance Distributed Computing HPDC* , (pp. 49-58).

i-MapReduce. (2009). Retrieved December 2009, from www.iterativemapreduce.org

Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2007). Dryad: Distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review. 41*, pp. 59-72. ACM Press.

JAligner. (2009). Retrieved December 2009, from Smith Waterman Software: http://jaligner.sourceforge.net

Kearsley, A. J., Tapia, R. A., & Trosset, M. W. (1995). *The Solution of the Metric STRESS and SSTRESS Problems in Multidimensional Scaling Using Newton's Method.* Houston, Tx: Rice University.

Klock, H., & Buhmann, J. M. (2000). Data visualization by multidimensional scaling: a deterministic annealing approach. *Pattern Recognition , 33*, 651-669.

Kohonen, T. (1998). The self-organizing map. *Neurocomputing , 21*, 1--6.

Kruskal, J. B., & Wish, M. (1978). *Multidimensional Scaling.* Sage Publications Inc.

Kruskal, J. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika , 29*, 1--27.

LINQ Language-Integrated Query. (2009). Retrieved December 2009, from http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

MacQueen, J. B. Some Methods for classification and Analysis of Multivariate Observations. *5-th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281-297). University of California Press.

Moab Cluster Tools Suite. (2009). Retrieved December 2009, from http://www.clusterresources.com/products/moab-cluster-suite.php

Moretti, C., Bui, H., Hollingsworth, K., Rich, B., Flynn, P., & Thain, D. (2009). All-Pairs: An Abstraction for Data Intensive Computing on Campus Grids. *IEEE Transactions on Parallel and Distributed Systems , 21*, 21-36.

MPI. (2009). Retrieved December 2009, from Message Passing Interface: http://www-unix.mcs.anl.gov/mpi/

Pallickara, S., & Fox, G. (2003). NaradaBrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. *ACM/IFIP/USENIX 2003 International Conference on Middleware.* Rio de Janeiro, Brazil: Springer-Verlag New York, Inc.

Pike, R., Dorward, S., Griesemer, R., & Quinlan, S. (2005). Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure vol. 13, no. 4* , 227–298.

Price, A. L., Eskin, E., & Pevzner, P. A. (2004). Whole-genome analysis of Alu repeat elements reveals complex evolutionary history. *Genome Res , 14*, 2245–2252.

PubChem project. (2009, December). Retrieved from PubChem project: http://pubchem.ncbi.nlm.nih.gov/

Qiu, X., & Fox, G. C. (2008). Data Mining on Multicore Clusters. *In Proceedings of 7th International Conference on Grid and Cooperative Computing GCC2008* (pp. 41-49). Shenzhen, China: IEEE Computer Society.

Qiu, X., Ekanayake, J., Beason, S., Gunarathne, T., Fox, G., Barga, R., et al. (2009). Cloud Technologies for Bioinformatics Applications. *2nd ACM Workshop on Many-Task Computing on Grids and Supercomputers (SuperComputing09).* ACM Press.

Qiu, X., Fox, G. C., Yuan, H., Bae, S.-H., Chrysanthakopoulos, G., & Nielsen, H. F. (2008). Performance of Multicore Systems on Parallel Data Clustering with Deterministic Annealing. *Computational Science – ICCS 2008* (pp. 407-416). Kraków, POLAND: Springer Berlin / Heidelberg.

ROOT, Data Analysis Framework. (2009). Retrieved December 2009, from http://root.cern.ch/

Rose, K. (1998). Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems. *Proceedings of the IEEE , 86*, 2210--2239.

Rose, K., Gurewitz, E., & Fox, G. (1990). A deterministic annealing approach to clustering. *Pattern Recogn. Lett. , 11*, 589--594.

Rose, K., Gurewitz, E., & Fox, G. C. (1990). Statistical mechanics and phase transitions in clustering. *Phys. Rev. Lett. , 65*, 945--948.

Salmon, J. K. (1991). *Parallel hierarchical N-body methods.* PhD. California Institute of Technology.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology , 147* (1), 195-197.

Takane, Y., Young, F. W., & de Leeuw, J. (1977). Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features. *Psychometrika , 42*, 7-67.

Thompson, B. (1984). *Canonical correlation analysis uses and interpretation.* Sage.

Wheeler, D. L., Barrett, T., Benson, D. A., Bryant, S. H., Canese, K., Chetvernin, V., et al. (2006). Database resources of the national center for biotechnology information. *Nucleic acids research* .

xCAT. (2009). *Extreme Cluster Administration Toolkit*. Retrieved December 2009, from http://xcat.sourceforge.net/

Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P., et al. (2008). DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. *Symposium on Operating System Design and Implementation (OSDI).*