Using MapReduce to Compare Large Collections of Phylogenetic Trees

Bryce Tyson
Bryce.Tyson@usma.edu

Rosemary Betros
Rosemary.Betros@usma.edu

Nathaniel Rollings
Nathaniel.Rollings@usma.edu

Jorge Figueroa-Cecco
Jorge.Figueroa-Cecco@usma.edu

Lisa Jones
Lisa.Jones@usma.edu

Suzanne J. Matthews[*]
suzanne.matthews@usma.edu

Department of Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996

ABSTRACT

Phylogenetic trees depict the relationships between sets of organisms sharing a common ancestor. In several recent court cases, phylogenetic trees have been critical in convicting and exonerating suspects of purposefully transmitting the HIV virus. Parallel computing applications such as MapReduce Speeds Up Robinson-Foulds (MrsRF) make it possible to compare large collections of phylogenetic trees efficiently, and can be used to ensure that the searches that produce these trees properly converged.

In this paper, we describe MrsRF++, an updated version of MrsRF that utilizes the Phoenix++ MapReduce engine. The original version of MrsRF was implemented with Phoenix 1.0, which had a number of critical limitations. Implementing MrsRF with Phoenix++ overcomes many of these shortcomings, and increased program modularity. We ran MrsRF++ on the same two large biological tree sets described in the original MrsRF paper. Our preliminary results indicate that using the Phoenix++ MapReduce engine increases MrsRF's scalability, a fact that will enable scientists to analyze increasingly large collections of phylogenetic trees.

Categories and Subject Descriptors

J.3 [**Life and Medical Science**]: Biology and Genetics; D.1.3 [**Software**]: Programming Techniques—*Parallel programming*

[*]Corresponding author.

General Terms

Design, Performance

Keywords

MrsRF++, MapReduce, HPC, Phylogeny, Computational Biology

1. INTRODUCTION

A phylogenetic tree depicts the relationships between a set of organisms (taxa) that share a common ancestor. Applications of phylogenetic trees include discovering the source of a virus [15] and concocting antivenins [5] and pharmaceuticals [4]. Recently, phylogenetic trees have been used as evidence in a number of criminal cases [1]. For example, in 2009, Philipe Padieu was convicted of purposely transmitting the HIV virus to multiple women. Prosecutors found the evidence to prove Padieu guilty by using a phylogenetic tree to trace the ancestry of the HIV strains found in his victims to the strain that he carried. For his crimes, he was sentenced to a total of 250 years in prison [11].

Scientists infer evolutionary trees using phylogenetic search algorithms, which uses heuristic techniques to search a massive $O(2n-5)!!$ space of hypothetical trees ("tree-space") and return statistically "good" trees. Scientists repeat the search multiple times to avoid locally optimal solutions. These analyses can produce hundreds of thousands of equally likely trees, which scientists summarize into a single tree called the consensus. They discard the majority of the source trees in the process.

However, the information discarded has shown to be useful for a variety of applications [8] [13], especially determining how successful a phylogenetic search is. If a phylogenetic search properly converged, then the trees produced from the multiple runs should be similar to each other [6]. In the absence of efficient methods for comparing the increasingly large collections of resulting trees, scientists are forced to assume that their searches converged. If these trees are accepted as evidence to convict people of crimes, however, it is critical to ensure that the searches that produced them converged. Parallel computing has great potential to allow scientists to compare phylogenetic trees more efficiently.

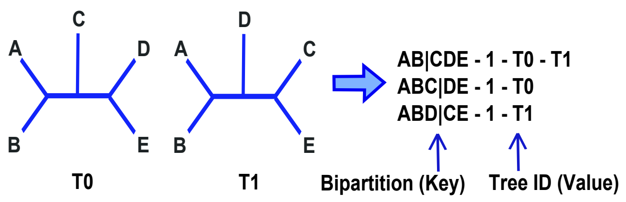

Figure 1: An illustration of bipartitions

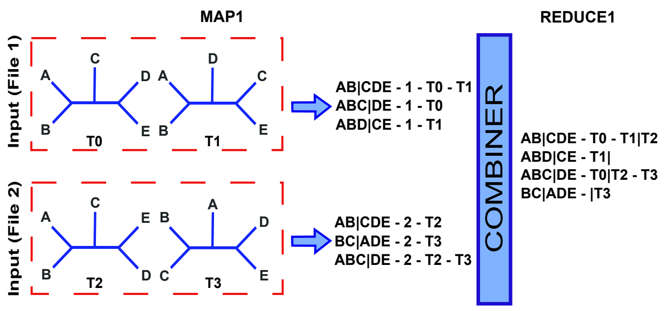

Figure 2: Phase 1 of MrsRF

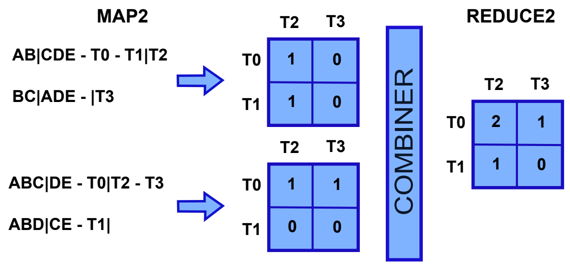

Figure 3: Phase 2 of MrsRF

Applications like Phoenix [9], for example, which uses the MapReduce [3] paradigm to allocate work across available nodes and cores, can significantly expedite the process.

2. RELATED WORK

MapReduce Speeds Up Robinson-Foulds (MrsRF) [8] is a parallel application capable of comparing large groups of phylogenetic trees very quickly. Through the use of universal hashing, it can quickly identify unique bipartitions in a collection of trees. A bipartition is a split along an internal edge of a tree that results in two independent sets of taxa. Consider tree $T0$ in Figure 1. Since $T0$ has two internal edges, it contains two bipartitions: AB|CDE and ABC|DE.

MrsRF uses the Robinson-Foulds (RF) distance [10], the most popular method of comparing trees. This metric expresses the topological distance between two probable phylogenetic trees based on common bipartitions. Consider tree $T1$ in Figure 1. $T1$ also contains two bipartitions: AB|CDE and ABD|CE. To compute the RF distance between trees $T0$ and $T1$, we use the following equation (where $B(T)$ is the set of bipartitions in tree T):

$$RF(T0, T1) = \frac{|B(T0) - B(T1)| + |B(T1) - B(T0)|}{2}$$

Notice that T0 and T1 share the bipartition AB|CDE but not ABD|CE or ABC|DE. Thus the RF distance between T0 and T1 is 1. To compare a collection of trees, MrsRF calculates a $t \times t$ RF matrix, where the value in cell (i,j) of the resulting matrix represents the distance between trees i and j.

MrsRF uses MapReduce [3], a popular programming paradigm designed to process large data sets. The framework allows programmers to easily create parallel computing applications. The programmer need only write a `map()` and `reduce()` function, and the rest of the process is automated. Due to the *(key, value)* nature of its data model, MapReduce is ideal for parallelizing algorithms that use hashing.

MrsRF is based on Phoenix [9], an open-source, shared memory implementation of MapReduce. The Phoenix software dynamically schedules tasks across available processors to maximize throughput and reduce overhead related to data communication and task spawning. Compared to other approaches (such as Hadoop [2]), implementing MrsRF with Phoenix led to major speed ups [8]. Phoenix does, however, have limitations, including its inefficient combiner implementation, poor task overhead amortization, and uniform intermediate storage of key-value pairs in hash tables [14]. These flaws cause the software to be inefficient for certain workloads.

Many of these limitations have been addressed in Phoenix++ [14], the most recent version of the software. Phoenix++ offers increased modularity, is written in C++, and uses templates to statically inline code without incurring significant performance penalties. It boasts increased scalability and is, on average, 4.7 times faster than its predecessor [14].

In order to ensure that the MrsRF algorithm can continue to keep pace with the growth of phylogenetic data, we wrote MrsRF++, an updated version of MrsRF that takes advantage of the Phoenix++ MapReduce library. MrsRF++ is written in C++, a change which enabled us to increase modularity and eliminate many function calls without performance penalties. For example while the original MrsRF was forced to run the subprogram HashBase as an executable, we can increase efficiency by integrating it directly into the MrsRF++ code.

3. APPROACH

MrsRF begins by partitioning the data over N nodes, each of which is responsible for computing an RF submatrix over the trees assigned to it. Each node then computes its assigned submatrix using independent instances of MapReduce. Each core executes two independent runs of MapReduce, which we refer to as Phase 1 and Phase 2. In the examples to follow, we illustrate how the algorithm creates the submatrix $\{T0, T1\} \times \{T2, T3\}$ using two cores. In Phase 1, the program creates a global hashtable of unique bipartitions and the trees that contain them. In Figure 2, File 1's trees $T0$ and $T1$ share bipartition AB|CDE while File 2's trees $T2$ and $T3$ share bipartition ABC|DE. The `map()` function extracts the bipartitions from each tree and emits them along with their tree identifier (TID), as the $(key, value)$ intermediates for Phase 1. The combiner groups common TIDs that belong to the same bipartitions, forming $(key, list(value))$ pairs. The `reduce()` function then combines the information together, resulting in the set of unique bipartitions and the TIDs from both files that contain them. In our example,

$T0$ and $T1$ from File 1 and $T2$ from File 2 are combined into a single list sharing the bipartition AB|CDE.

In Phase 2, the program creates similarity matrices and computes the final RF distance matrix. In Figure 3, the input to the `map()` function is the set of unique bipartitions and their associated TIDs. This set is split into pieces, which each instance of `map()` uses to construct a local similarity matrix. If a set of trees shares a common bipartition, `map()` increments the value in the appropriate matrix box by 1. In Figure 3, cell $(T0, T2)$ is incremented by 1 because both T0 and T1 share bipartition AB|CDE. Next, the `reduce()` function combines each row of each similarity matrix and adds up the corresponding boxes. Since the independent instances of the `map()` function updated the cell $(T0, T2)$ twice, the similarity matrix produced by `reduce()` contains the value 2. Lastly, the similarity matrix is used to compute the RF distance matrix.

4. PRELIMINARY RESULTS

We used two biological data sets to test the performance of MrsRF++: a collection of 150 taxa and 20,000 trees [7] and a larger collection over 567 angiosperm taxa and 33,306 trees [12]. We note that these are the same two biological datasets originally used to ascertain the performance of MrsRF. The 150 taxa dataset contains $1,128$ unique bipartitions, while the 567 taxa dataset contains $2,444$ unique bipartitions.

All of our experiments were run on a 64-bit machine with two quad-core processors (8 total cores) and 32 GB of RAM. Our single node cluster runs Ubuntu Linux 12.04. Both MrsRF++ and MrsRF were compiled with `gcc` 4.6.3 and the `-O3` compiler flag. We ran each program three times on both datasets over 1, 2, 4, and 8 cores. Running time is reported as the average over all three runs. We calculated speedup with the equation

$$SpeedUp = \frac{T(1 \text{ core})}{T(n \text{ cores})}$$

where $T(1 \text{ core})$ is the execution time of MrsRF++ on a single core, and $T(n \text{ cores})$ is the execution time of MrsRF++ when run over n cores.

We first compared the performance of MrsRF++ to its predecessor, MrsRF. We ran MrsRF and MrsRF++ on both biological datasets over 1, 2, 4, and 8 cores. In all these cases, MrsRF++ is only slightly faster than MrsRF. For example, when run on the 150 taxa dataset, MrsRF took 37.61 seconds on 1 core, while MrsRF++ took 37.28. When run with 8 cores on this dataset, MrsRF took 9.86 seconds, while MrsRF++ took 9.32 seconds. On the 567 taxa dataset, MrsRF took 401.93 seconds on 1 core, while MrsRF++ took 397.28 seconds. On 8 cores for this dataset, MrsRF took 62.81 seconds, while MrsRF++ took 51.44 seconds. Since MrsRF++ has the fastest serial implementation, all speedups reported use MrsRF++'s time on a single core.

While Phoenix++ is up to 4.7 times faster than its predecessor, we believe those performance gains are largely lost in MrsRF++ due to the unique nature of its workflow. Most applications of MapReduce have a larger input dataset which gets processed and reduced to a smaller output. However, MrsRF and MrsRF++ produce a much larger output (the $t \times t$ RF matrix) than the input set of t trees.

4.1 Overall Performance

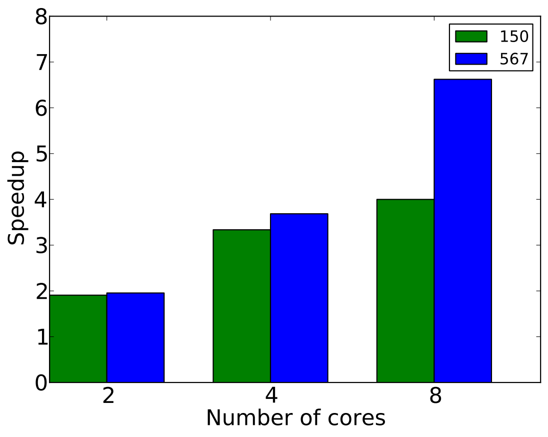

Figure 4: MrsRF++ speedups on 2, 4, and 8 cores. Speedup is computed with respect to MrsRF++ on 1 core.

In an ideal scenario, when MrsRF++ is run on n cores, it will result in a speedup of n. Figure 4 shows the overall performance of MrsRF++ on n cores with respect to MrsRF++ on 1 core. The 150 taxa dataset took on average 37.28 seconds to run on 1 core, and 9.32 seconds when run on 8 cores. The 567 taxa dataset took on average 397.28 seconds on 1 core, and 59.99 seconds when run on 8 cores.

On the 150 taxa $20,000$ trees dataset, MrsRF++ achieves a near perfect speedup of 1.91 on 2 cores. When we ran MrsRF++ on this dataset over 4 and 8 cores, we achieved speedups of 3.34 and 4.00 respectively. We hypothesize that the reduction of performance over higher numbers of cores is due to the fact that there simply is not enough data to fully utilize 4 and 8 cores. On 4 cores, MrsRF++ takes on average 11.18 seconds to compute the RF matrix; in other words, running MrsRF++ on 8 cores on this dataset results in average improvement of only 1.86 seconds.

Our experimentation on the larger, 567 taxa and $33,306$ tree dataset lends credence to this hypothesis. Once again, MrsRF++ achieves near-perfect speedup of 1.95 on 2 cores. Since this dataset is larger, speedups increase to 3.68 and 6.62 on 4 and 8 cores. While it took MrsRF++ on average 107.81 seconds to process this dataset on 4 cores, running time was reduced to 59.99 seconds on 8 cores.

4.2 Phase 1 and 2 Performance

To investigate potential bottlenecks in the performance of MrsRF++ we also conducted a separate analysis of Phase 1 and Phase 2. Figure 5 shows the performance on Phase 1 of MrsRF++ on the 150 and 567 taxa datasets. In general, we obtained fairly good speedups for Phase 1. On the 150 taxa dataset, MrsRF++ took 8.12 seconds to execute on 1 core. Speedup increased from 1.92 on 2 cores to 3.3 and 5.65 on 4 and 8 cores respectively. This corresponds to running times of 4.22, 2.46, and 1.44 seconds, respectively, for 2, 4 and 8 cores.

On the 567 taxa dataset, MrsRF++ took 54.88 seconds during Phase 1 on a single core. For 2 cores, we obtained speedup of 1.92 (28.57 seconds), which increased to speedups

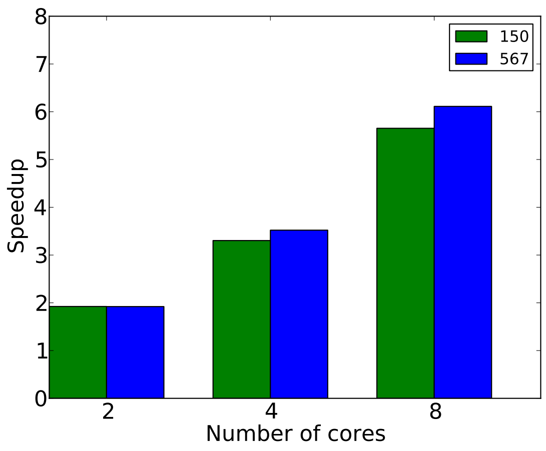

Figure 5: Phase 1 speedups on 2, 4, and 8 cores. Speedup is computed with respect to MrsRF++ Phase 1 on 1 core.

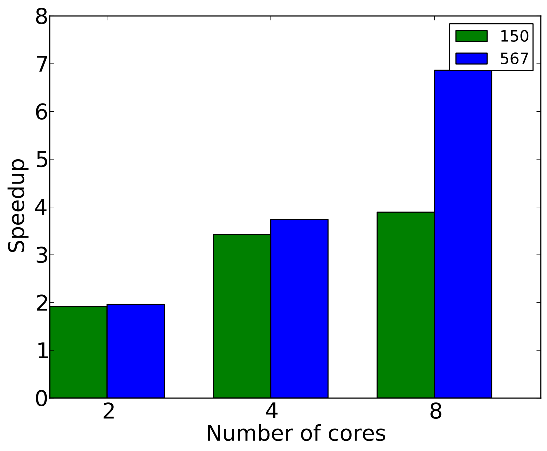

Figure 6: Phase 2 speedups on 2, 4, and 8 cores. Speedup is computed with respect to MrsRF++ Phase 2 on 1 core.

of 3.52 (28.57 seconds) and 6.11 (8.98 seconds) on 4 and 8 cores respectively. For both datasets, the majority of the time is spent on the extraction of bipartitions from the source trees. A reduction in the time needed to perform this step will likely improve the speedups gained by Phase 1 of MrsRF++.

Figure 6 illustrates the Phase 2 performance of MrsRF++ over our two biological datasets. For 150 taxa, Phase 2 of MrsRF++ takes 28.98 seconds on 1 core. On 2 cores, we obtain a speedup of 1.91 (15.15 seconds), which increases to 3.43 (8.45 seconds) and 3.89 (7.44 seconds) on 4 and 8 cores, respectively. We strongly believe this decrease in speedup on this dataset is due to the algorithm lacking enough data to fully take advantage of 8 cores. For Phase 2 of the algorithm, "data" is the number of unique bipartitions contained in the source set of trees. For the 150 taxa set, there are only 1,128 unique bipartitions over 20,000 trees.

Our results on the 567 taxa datasets support this hypothesis. This larger dataset has 2,444 unique bipartitions over 33,306 trees. On 1 core, Phase 2 of MrsRF++ takes 342.0 seconds. When we increase the number of cores to 2, we gain a near-perfect speedup of 1.96 (174.14 seconds). Increasing the number of cores again to 4 results in a speedup of 3.74 (91.48 seconds). However, once we get to 8 cores, we encounter some performance degradation, as speedup is reduced to 6.86 (49.82 seconds). We believe this is again due to the algorithm having too few bipartitions to process on 8 cores.

5. CONCLUSIONS

In this paper, we introduce MrsRF++, an updated implementation of the MrsRF algorithm for comparing large groups of phylogenetic trees. As phylogenetic tree collections continue to grow, they become increasingly computationally expensive to compare. To keep pace with changing times, we performed a critical update to the MrsRF software to utilize the Phoenix++ MapReduce engine. Phoenix++ addresses many of the shortcomings of its predecessor, Phoenix 1.0, allowing for the creation of more modular and scalable code.

We tested MrsRF++ on the two biological datasets originally used to benchmark MrsRF. Our preliminary results indicate that MrsRF++ and Phoenix++ appear to more efficiently compare large sets of phylogenetic trees. Our results suggest that MrsRF++ is not only faster than MrsRF, but achieves better scalability than its predecessor. While our implementation experienced performance degradation on 8 cores, we believe this is due to a lack of data, and not any fault on the part of the program. We hypothesize that as we obtain and benchmark larger datasets, MrsRF++'s core utilization on 4 and 8 cores will only continue to improve.

For the future, we plan on fine-tuning the performance of MrsRF++. Our results are still very preliminary; more extensive analysis is needed to fully explore the utility of MrsRF++. To this end, we plan on benchmarking MrsRF++ over larger sets of trees and on a large multi-node cluster. Like its predecessor, MrsRF++ is currently designed for unweighted, binary trees. In our next iteration, we also plan on extending MrsRF++'s functionality to include more diverse collections of trees.

6. ACKNOWLEDGEMENTS

This project is largely funded by a 2013-2014 Collaborative Research Experiences for Undergraduates (CREU) award, sponsored by the Computing Research Association's Committee on the Status of Women in Computing Research (CRA-W) and the Coalition to Diversify Computing (CDC). The first five authors are undergraduate student researchers. The last author is the faculty advisor. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the U.S. Military Academy, or the U.S. Army.

7. REFERENCES

[1] E. J. Bernard, Y. Azad, A.-M. Vandamme, M. Weait, and A. M. Geretti. Hiv forensics: pitfalls and acceptable standards in the use of phylogenetic analysis as evidence in criminal investigations of hiv transmission*. *HIV medicine*, 8(6):382–387, 2007.

[2] D. Borthakur. The hadoop distributed file system: Architecture and design, 2007.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the Association of Computing Machinery*, 51(1):107–113, January 2008.

[4] X.-X. Gao, H. Zhou, D.-Y. Xu, C.-H. Yu, Y.-Q. Chen, and L.-H. Qu. High diversity of endophytic fungi from the pharmaceutical plant, heterosmilax japonica kunth revealed by cultivation-independent approach. *FEMS Microbiology letters*, 249(2):255–266, 2005.

[5] N. Giannasi, R. S. Thorpe, and A. Malhotra. The use of amplified fragment length polymorphism in determining species trees at fine taxonomic levels: analysis of a medically important snake, trimeresurus albolabris. *Molecular Ecology*, 10(2):419–426, 2001.

[6] J. P. Huelsenbeck, B. Larget, R. E. Miller, and F. Ronquist. Potential applications and pitfalls of bayesian inference of phylogeny. *Systematic biology*, 51(5):673–688, 2002.

[7] L. A. Lewis and P. O. Lewis. Unearthing the molecular phylodiversity of desert soil green algae (chlorophyta). *Systematic Biology*, 54(6):936–947, 2005.

[8] S. J. Matthews and T. L. Williams. MrsRF: an efficient mapreduce algorithm for analyzing large collections of evolutionary trees. *BMC Bioinformatics*, 11(Suppl 1):S15, 2010.

[9] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *IEEE 13th International Symposium on High Performance Computer Architecture (HPCA'07)*, pages 13–24, Feb. 2007.

[10] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

[11] D. I. Scaduto, J. M. Brown, W. C. Haaland, D. J. Zwickl, D. M. Hillis, and M. L. Metzker. Source identification in two criminal cases using phylogenetic analysis of HIV-1 DNA sequences. *Proceedings of the National Academy of the Sciences (PNAS)*, 107(50):21242–21247, 2010.

[12] D. E. Soltis, M. A. Gitzendanner, and P. S. Soltis. A 567-taxon data set for angiosperms: The challenges posed by bayesian analyses of large data sets. *International Journal of Plant Sciences*, 168(2):137–157, 2007.

[13] S.-J. Sul, S. J. Matthews, and T. L. Williams. Using tree diversity to compare phylogenetic heuristics. *BMC Bioinformatics*, 10(Suppl 4):S3, 2009.

[14] J. Talbot, R. Yoo, and C. Kozyrakis. Phoenix++: Modular mapreduce for shared-memory systems. In *In the Second International Workshop on MapReduce and its Applications (MAPREDUCE)*, 2011.

[15] R. G. Webster, W. J. Bean, O. T. Gorman, T. M. Chambers, and Y. Kawaoka. Evolution and ecology of influenza a viruses. *Microbiological reviews*, 56(1):152–179, 1992.