LBE: A Computational Load Balancing Algorithm for Speeding up Parallel Peptide Search in Mass-Spectrometry based Proteomics

Muhammad Haseeb†, Fatima Afzali†, Fahad Saeed*†
†School of Computing and Information Sciences
Florida International University, Miami, Florida 33199, USA
Email: {mhaseeb,aafza002,fsaeed}@fiu.edu

***Abstract*—The most commonly employed method for peptide identification in mass-spectrometry based proteomics involves comparing experimentally obtained tandem MS/MS spectra against a set of theoretical MS/MS spectra. The theoretical MS/MS spectra data are predicted using protein sequence database. Most state-of-the-art peptide search algorithms index theoretical spectra data to quickly filter-in the relevant (similar) indexed spectra when searching an experimental MS/MS spectrum. Data filtration substantially reduces the required number of computationally expensive spectrum-to-spectrum comparison operations. However, the number of predicted (and indexed) theoretical spectra grows exponentially with increase in post-translational modifications creating a memory and I/O bottleneck. In this paper, we present a parallel algorithm, called LBE, for efficient partitioning of theoretical spectra data on a distributed-memory architecture. Our proposed algorithm first groups the similar theoretical spectra. The groups are then finely split across the system allowing machines to perform almost equal amount of work when querying a MS/MS spectrum. Our results show that the compute load imbalance using LBE based data distribution is $\leq 20\%$ allowing speedups of order of magnitudes over existing methods. The proposed algorithm has been implemented on a compute cluster using MPI library. Experimental results for increasing index sizes are reported in terms of execution time, speedups and memory footprint. To the best of our knowledge, LBE is the first load-balancing technique for MS/MS proteomics data on memory-distributed clusters that incorporates proteomics domain knowledge for efficient load-balancing. Source code is made available at: https://github.com/pcdslab/lbdslim/tree/mpi**

***Index Terms*—proteomics, mass-spectrometry, load balancing, indexing, distributed computing**

I. Introduction

Fast and accurate identification of peptides in an experimental MS/MS spectra dataset is a critical bottleneck in proteomics research since the rate of experimental data generation is much higher than its processing and identification rate [1]. The experimental MS/MS data are generated from a shotgun proteomics experiment which involves enzymatic proteolysis of a complex protein mixture followed by an automated liquid-chromatography (LC) followed by mass-spectrometry (LC-MS/MS) pipeline [2]. The obtained tandem MS/MS spectra are then computationally compared against a set of theoretically predicted MS/MS spectra [1] , [3] (also referred to as reference data). The theoretical MS/MS spectra are generated using the peptide sequences obtained by in-silico digestion of a protein sequence database. The in-silico methods for peptide and spectra generation allow fine control over the characteristics (post-translational modifications, peptide lengths etc.) and volume of generated reference data.

Both the set of experimental (query) and the predicted (reference) spectra contain millions of spectra each [4]. Therefore, computing similarity scores for all experimental spectra against all reference spectra is highly inefficient and extremely compute intensive. Therefore, a two-step approach is employed where the reference data are first pre-indexed which allows reference data filtration such that the number of spectra-to-spectrum comparisons between are substantially reduced. Several database filtration methods have been introduced that index the reference data (peptides and spectra) to quickly filter the reference data that have high likelihood of correctly matching to a given query spectrum. However, there are two caveats associated with peptide data filtration. First, the size of index may grow much larger than available memory resources on the system. Second, parallel querying large index at high frequency bounds the performance by I/O bandwidth bottleneck.

Motivation: This huge demand for CPU, I/O and memory resources is a bottleneck for the performance of existing state-of-the-art peptide search algorithms including [5], [6], [7], [8], [9]. Therefore, we propose employment of distributed systems for peptide identification problem such that the data and workload are spread over a set of independent asynchronous compute nodes to alleviate resource bounds. However, to achieve maximum system throughput, the data must be efficiently partitioned to ensure load balance across the system. To the best of our knowledge, there is no existing mechanism to partition theoretical MS/MS data such that the query-load is balanced across the parallel architecture. Further, the methods designed for shared memory systems cannot be directly used for distributed system since they would result in imbalanced system and high communication costs.

Contributions: In this paper, we present a parallel algorithm, called LBE, for distributing reference data across a set of symmetrical parallel compute units so that the querying load is almost equal across the system units. The proposed algorithm first collects highly similar reference data elements

* Corresponding Author

into groups. The data clustering/grouping method depends on the underlying algorithm employed for data filtration. The grouped data are finely spread among the parallel units such that each compute node has a similar data distribution. In this paper, we focus on data partitioning with respect to *shared-peak* based data filtration methods. We implemented our algorithm in the SLM-Transform index [6] (also called SLM-Index) code base. We chose SLM-Index because it is the most memory-efficient among other state of the art *shared-peak* based algorithms. Further, its source code is open-sourced unlike other algorithms. The proposed algorithm has been implemented on a cluster of workstations using MPI library. Our results show that the system imbalance for LBE based data distribution is under 20% enabling significant performance speedup over conventional partitioning method. The source code of *LBDSLIM* is available as open-source software at: https://github.com/pcdslab/lbdslim/tree/mpi

II. Related Work

A. Data Filtration Methods

*1) **Peptide Precursor Mass**:* or simply peptide mass based filtration method restricts the search space to only the reference peptides having precursor mass within query spectrum's mass $\pm$ the precursor mass tolerance window. However, the precursor mass based filtration fails to identify the query spectra that contain unknown post-translational modifications. i.e. 'the dark Matter of shotgun proteomics' [10], [11], [12]. As a solution, the precursor mass tolerance is usually increased to several hundred Daltons (open-search) but that results in search spaces that too huge to be efficiently processed [5], [13].

*2) **Sequence-Tag**:* based search space filtration method extracts a few amino acid characters long sequence tags from each query spectrum and the search space is restricted to the peptide sequences generated from all database protein sequences containing one more k-mers of the extracted sequence-tag. The variants of *sequence-tag* method has been employed in [14], [7], [15], [16], [17].

*3) **Shared Peak Count**:* or *fragment-ion* based filtration method indexes all the ions in reference spectra along with their peptide sequences. The search space is then restricted to only the indexed reference spectra that share more than a certain number of ions, also referred to as peaks or fragments, with the query spectrum. The variants of *shared-peak* based data filtration method has been employed in [18], [19], [20], [5], [21], [22].

B. Shared Memory Data Partitioning

Since the size of index grows exponentially and usually beyond available main memory on the system, the index must be partitioned into smaller chunks. The chunks are processed independently and may be stored on disks when not in use. The existing shared memory data distribution methods group the similar data by first sorting the peptide entries in the index by their precursor masses. Then the ion-index is sorted by first ion-masses and then their parent peptide masses. The index is then simply partitioned into smaller independent chunks as shown in Fig. 1. The data partitioning scheme ensures that for a given experimental spectrum, the similar reference data are located as contiguous entries at only *one* index chunk. This technique speeds up the search by reducing the number of index chunks that need to be loaded into memory or processed along with superior cache performance when accessing data for further processing.

Employing existing partitioning method for distributed system based partitioning may result in some system machines to share all computational load while rest of the machines are idling resulting in an inefficient system as depicted in Fig. 2.

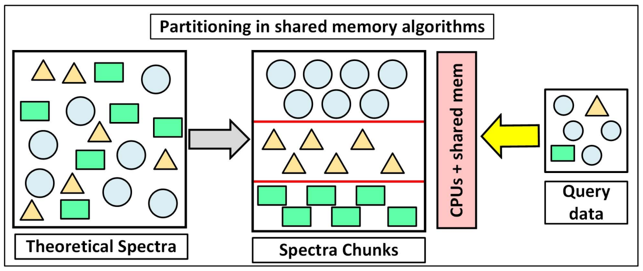

Fig. 1. The reference spectra in shared memory algorithms are sorted and then partitioned into chunks so that the similar data may lie as contiguous entries on the same chunk.

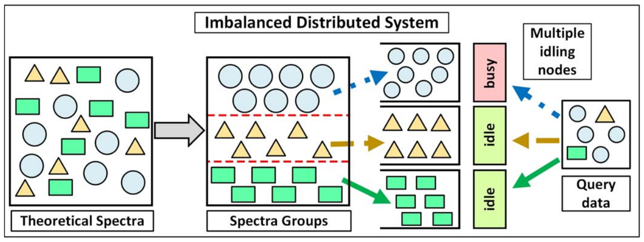

Fig. 2. The existing method if employed for partitioning across distributed memory system will result in groups of similar spectra to be located on one machine. This may result in highly imbalanced system when querying the index.

III. Methods

A. Problem Statement

The distributed memory based data distribution requires data partitioning such that for a given experimental spectrum, equal number of similar reference spectra may lie as contiguous entries on ***all*** machines in the system as shown in Fig. 3.

B. The LBE Method

The LBE algorithm design ensures the system load balance by sorting/clustering the reference data into groups. The groups are then partitioned across the system such that data sketch is similar across the system as illustrated in Fig. 3. To minimize communication among nodes, the peptide information is stored only on the master machine while all the machine work with their own indices and only the results are transferred to main machines which are mapped back to original entries via the mapping table. The subsequent sections discuss the methods employed by LBE in detail.

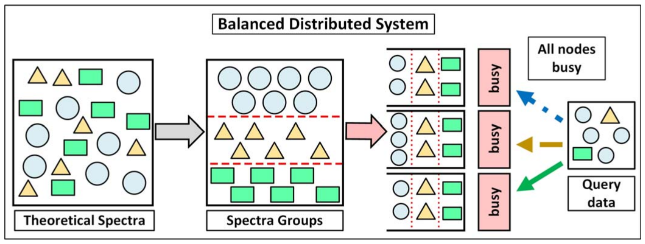

Fig. 3. The LBE partitioning method first sorts the theoretical spectra into groups which are finely partitioned to allow similar data sketch across all system machines. The data may be further partitioned at each node according to the scheme shown in Fig. 1.

C. Data Grouping

The employed grouping method depends on the underlying database filtration algorithm. For instance, if the underlying algorithm filters reference data based on *precursor masses*, then the LBE must ensure identical average peptide precursor mass across the system. In our case i.e. *shared peak count* based filtration, the peptide sequences are clustered since similar peptide sequences yield similar theoretical spectra as shown in Fig. 3. We implemented a data grouping method that is similar to [23]. The existing amino acid sequence clustering algorithms [24], [25], [26], [27] can also be used. The purpose of data grouping/clustering is to localize the peptide sequences that will yield similar theoretical spectra so that they could be equally spread across the system. Further, our method does not consider the variant peptide grouping and the normal peptide sequences and their modified variants are considered to be part of the same data group. The modified variant theoretical spectra may be very different if they have multiple modifications or even single modification at or near either N- or C- terminus of peptide sequence. Therefore, the MS/MS spectra clustering algorithms such as [28], [9], [29], [30], [12] may be employed in future to cluster the reference data at spectra level instead of peptide sequence level. The data grouping algorithm employed by LBE is explained as follows:

*1) **Grouping Peptide Sequences**:* The peptide sequences are first sorted by their lengths followed by lexicographical sort. Then the peptide groups are formed in the following manner. Start the first group g_1 from the first peptide sequence s_1. Then the algorithm looks for subsequent peptide sequence entries $s_j \mid 1 < j < i + gsize$ (def: $gsize = 20$) that satisfy $EditDistance(s_1, s_j) \leq max\{d, len(s_j)/2\}$ (def: $d = 2$) to be included in g_1. We also employed another grouping criterion given as: $EditDistance(s_1, s_j)/max\{len(s_1), len(s_j)\} \leq d'$ (def: $d' = 0.86$). The parameters d or d', grouping criteria, and $gsize$ may be configured as required. Moving on, the next group g_2 starts from the next peptide sequence in the database and the same process is repeated until all sequences are grouped as depicted in Algorithm 1.

*2) **Output**:* The peptide sequences contained in each group are concatenated together along with rest of the groups in FASTA format to yield a clustered database.

Algorithm 1: Grouping Peptide Sequences

```
Data: List of peptide sequences ($Li$
) Result: List of group sizes ($Lz$)
/* SortByLength                              */
$Li.SortByLength()$;
/* Lexicographical Sort                      */
$Li.LexSort()$;
/* First sequence in group                   */
$seq = Li[1]$;
/* List: group sizes                         */
$Lz.Append(1)$;
for $k = 2$ to $size(Li)$ do
    /* Edit Distance                          */
    $dist = EditDistance(seq, Li[k])$;
    /* Any of the cutoffs may be used
        */
    $cutoff_1 = max\{d, len(Li[k])/2\}$;
    $cutoff_2 = dist/max\{len(seq), len(Li[k])\} \leq d'$
    /* Check conditions                       */
    if ($dist > cutoff_x$) OR ($Lz[size(Lk)] == csize$)
     then
        /* Init new group                     */
        $seq = Li[k]$;
        $Lz.Append(1)$;
    else
        /* Increase curr group size           */
        $Lz[size(Lk)] += 1$;
    end
end
```

D. Data Partitioning

The clustered data are read by all system machines where each machine extracts out its data partition according to the specified distribution policy. Please note that the data partitioning is mandatory if the index size is larger than 2 billion ions (8GB) since C++ arrays are indexed by data type **int**. The extracted data are then processed by the SLM-Transform [6] to construct parallel partial SLM-Index on each machine. Note that the data are re-grouped and may require further partitioning by SLM-Transform at each machine as shown in Fig. 3. The MPI master machine constructs a mapping table to backtrack each machine's peptide indices to original peptide index entries. The mapping table is a simple an array of size N (size of peptide index) where each i^{th} chunk of array of size N/p contains the indices of peptide index entries mapped to machine i. The rest of compute machines discard their partial peptide indices after construction of SLM-Index. Assuming total number of entries in SLM-Index to be N, and the number of machines in the system to be p, each machine roughly receives N/p peptide entries. The distribution policies available in LBE are discussed as follows:

*1) **Chunk**:* policy splits the input data in simple chunk fashion. Chunk policy is equivalent to conventional partitioning method that splits the whole input data into chunks. Assuming the total number of entries in input is N, the number of machines in the system to be p, the peptides $pep(m)$ indexed by machine m using *Chunk* policy are given as:

$$pep(m) = \{i \mid \frac{N}{p} \times m \leq i < \frac{N}{p} \times (m+1)\}$$

*2) **Cyclic**:* policy distributes the peptide sequences in each group in a round robin fashion. Assuming the total number of entries in input is N, the number of machines in the system to be p, the peptides $pep(m)$ indexed by machine m using *Cyclic* policy are given as:

$$pep(m) = \{i \mid \wedge (i \mod m = 0)\}$$

*3) **Random**:* policy distributes the peptide sequences in each group in a random fashion. The peptide sequences in each group are shuffled and split using the *Chunk* policy. The quality of distribution may depend on initial choice of seed value. Assuming the total number of entries in input is N, the number of machines in the system to be p, the peptides $pep(m)$ indexed by machine m using *Random* policy are given as:

$$pep(m) = \{chunk(shuffle(i))\}$$

E. Distributed Querying

The raw MS/MS data are converted to mzML or MS2 format using msconvert.exe [31] before any processing. Then, all compute units in the system read the query spectra from the MS/MS dataset and pre-process them using the specified fragment extraction parameters. The compute units search the query spectra concurrently against their partial SLM-Index chunks. The peptide entries for resultant peptide-to-spectrum match candidates can then be returned to the user at MPI master machine by simply sending virtual indices to the master machine where they are mapped back to original peptide entries in $O(1)$ time (simple 1 memory access). This is illustrated in Fig. 4.

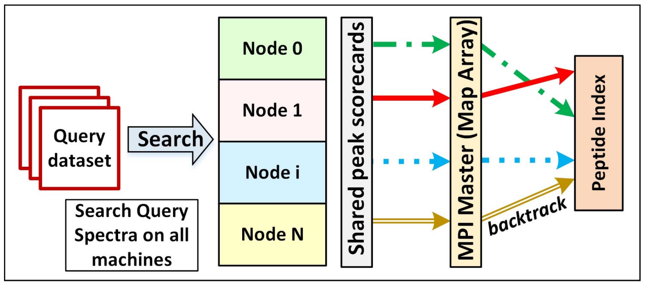

Fig. 4. The data are searched by each machine against its partial index. The indices of matched peptides/spectra are sent to MPI master where they are mapped to original peptide index entries in $O(1)$ time using the mapping array.

IV. Implementation

The LBE algorithm has been implemented as an abstraction layer, called *LBE layer*. The SLM-Transform code base was modified to make it compatible with MPI (mpi branch of LBDSLIM GitHub repository). The peptide sequence data grouping explained in the text has been implemented separately as a Python 3.x script (provided with source code) that prepares the input data for LBE based distributed SLM-Transform. This will be incorporated in the main code base in the future versions. The software can be compiled using GCC toolchain + MPI distribution on a Linux machine (GCC 6.3.0 + MPICH/OpenMPI) or Windows machine (MinGW GCC 6.3.0 + Microsoft MPI). We also implemented another version of software (ircc branch of LBDSLIM GitHub repository) to make the codebase compatible with GCC 4.6.2 (without glibc v2.14) + OpenMPI 1.6.3 to allow its use on computers/clusters that have with old compiler versions. The build is managed by a simple Makefile script. Please refer to the README on GitHub repository for detailed information about how to build, configure and use the software as well as any limitations or known issues in software.

V. Results

A. Experimental Setup

1) ***Database Preprocessing***: The Human proteome database (UP000005640) was downloaded from UnitProtKB. The database was digested in-silico using Digestor [32] tool which yielded a peptide sequence database. The digestion settings used are as follows: fully tryptic, upto 2 missed cleavages, peptide lengths from 6 to 40, peptide mass from 100amu to 5000amu. Following digestion, the duplicate peptide sequences were then removed using DBToolkit [33]. The peptide sequence data were clustered using criterion 2 with default settings using Algorithm. 1.
2) ***Dataset Preprocessing***: The dataset PXD009072: Changes in the human platelet protein ubiquitylation landscape following GPVI activation, containing 305,431 query spectra was downloaded from PRIDE Archive. The dataset files were converted to MS2 format using msconvert.exe [31].
3) ***SLM-Transform settings***: The SLM-Transform was configured to extract the 100 most intense peaks from each query spectrum. The rest of SLM-Transform settings used are as follows: Internal Data partitioning = *Disabled*, Resolution ($r = 0.01$), fragment mass tolerance ($\Delta F = 0.05Da$), precursor mass tolerance ($\Delta M = \infty$), Shared-Peak threshold ($Shpeak \geq 4$), and max modified residues per peptide to 5. The variable modifications including deamidation on asparagine and glutamine, gly-gly adducts on lysine/cysteine, and oxidation on methionine residues were added to the index. The parameter values were chosen based on the description of dataset on PRIDE website.
4) ***Compute Cluster***: The experiments were conducted on an HPC compute cluster. The allowed access to cluster resources was restricted to 4 physical machines, 16 cores and 32 GB RAM so we could not go beyond the allowed resources. Please note that the employed CPUs were symmetrical or nearly symmetrical.

The complete dataset search yielded 22,517,426,929 candidate peptide-to-spectra matches (cPSMs) ($\sim$ 73,723 cPSMs/query). We benchmarked the LBE based distributed SLM-Index for multiple performance metrics by searching the FL0320_MSQ805_IBombik_Stimb_6ul.ms2 file (23,264 spectra) from PXD009072 dataset against distributed SLM-index instances with variable sizes and executed on different cluster resource configurations.

B. Memory Footprint

The memory footprint for distributed SLM-Index implementation was compared against its shared memory implementation for increasing index size and number of machines in the system. The index size was varied by changing the type and number of amino acid modification settings. The memory footprint results for distributed implementation of SLM-Index show an average memory footprint of 0.366GB/million spectra indicating only 6.4% extra memory overhead memory footprint reported for shared memory implementation under the same settings (i.e. 0.346GB/million spectra). The extra memory overhead varies inversely with the size of data partition per MPI CPU. Please note that there is a temporary memory footprint of 100% for distributed SLM-Transform index (requires 2$\times$ index memory) as compared to shared memory implementation. The temporary overhead can be eliminated if internal index partitioning is allowed. However, this temporary requirement limits the index partition size per MPI process to 10.5 million spectra. Fig. 5 shows the comparison between memory footprint for distributed memory SLM-Transform against its original shared memory implementation.

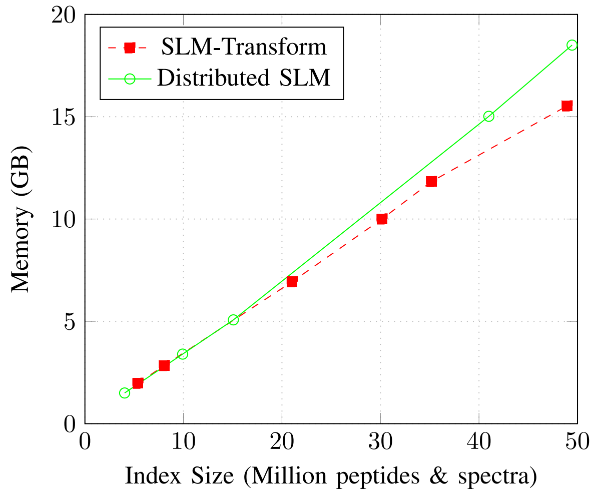

Fig. 5. The results show a small memory overhead for distributed SLM-Transform index over its original shared memory implementation.

C. Load Balance

We evaluated the load imbalance for all distribution policies for increasing index size. The experiments were performed with 16 MPI processes (4 physical machines) therefore, 16 index partitions. Assume that the average time consumed by system machines for an experiment is T_{avg}. The load imbalance would be zero if the time consumed by all system machines run in T_{avg}. We define the Load Imbalance (LI) as the maximum positive deviation in compute time (ΔT_{max}) normalized to average/ideal compute time T_{avg} given as:

$$LI = \frac{\Delta T_{max}}{T_{avg}} \tag{1}$$

The results obtained for Load Imbalance for cyclic and random distribution show a substantially superior system performance as compared to shared memory system like (chunk) data partitioning as shown in Fig. 6. The normalized Load Imbalance LI factor represents the relative system stall due to imbalance where high imbalance seriously hampers the system throughput. The detailed results for the experiments are available at LBDSLIM GitHub repository (mpi and ircc branch).

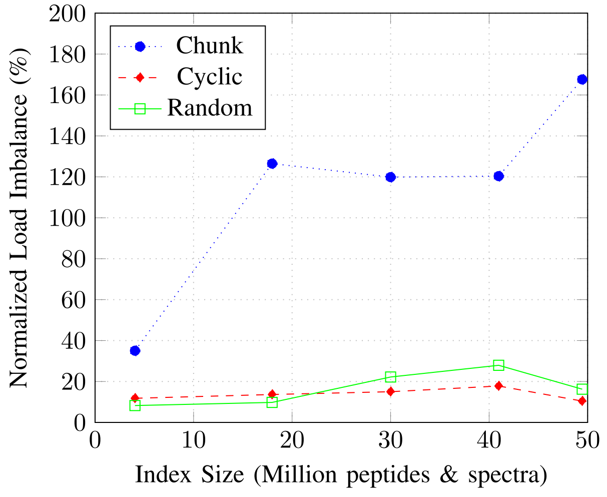

Fig. 6. This plot depicts percentage LI for 16 MPI CPUs (16 partitions) with increasing index size. The results show that the load imbalance remains $\leq$ 20% for cyclic and random distribution as compared to $\sim$ 120% imbalance for conventional (chunk) data partitioning.

D. Scalability

The speedup scalability was evaluated for both distributed SLM-Index querying and total execution time. The experiments were performed for increasing number of MPI processes repeated for increasing index sizes. The index was partitioned using *cyclic* distribution in all experiments since it allows the best load balance. All experiments were run on at least 2 physical machines. The results discussed in this section incorporate any speedup degradation due to load imbalance. Fig. 7 shows the query time results while Fig. 8 shows the speedup scalability for query time. Fig. 8 shows that the query time scales almost linearly as the number of CPUs are increased. Since, the allowed CPU resources were limited to 16 as well as the memory resources to 32GB, we could not perform experiments beyond 16 MPI processes and 49.5Million spectra index size. However, the results indicate a linear scalability as the system resources increase.

We also evaluated the speedup scalability for total execution time as shown in Fig. 9 and Fig. 10. Fig. 10 indicates that the total execution time does not scale linearly and saturates. The speedup saturation is due to the serial part of software. The base case (ideal speedup efficiency) used for computing speedup results in Fig. 8 was 2 CPUs for 18M index size and 4 CPUs for the rest. The average efficiency for experiments with higher number of CPUs was computed and used to scale that set of experiments including the base case. The detailed scalability results are available with LBDSLIM GitHub repository (mpi and ircc branch).

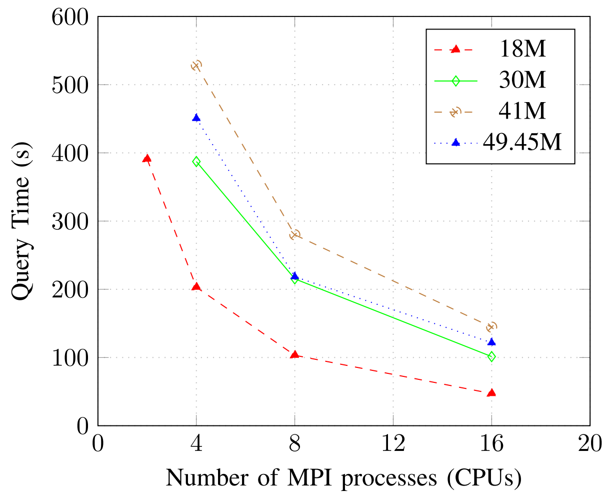

Fig. 7. This plot shows the query time results for cyclic partitioning with increasing number of MPI processes (CPUs) with increasing index size.

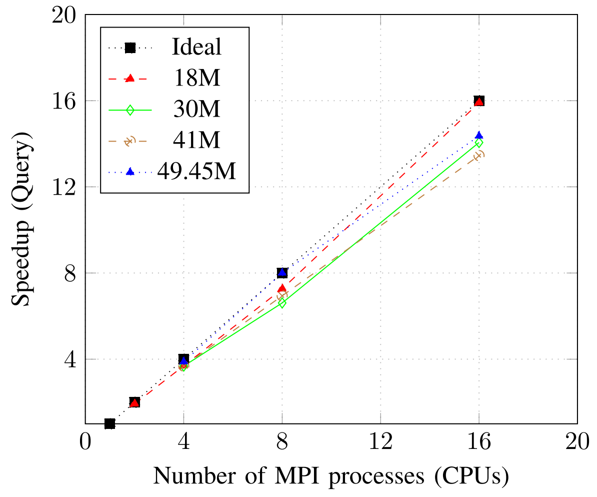

Fig. 8. The speedup results for distributed querying using cyclic partitioning show linear scalability. The results for 1 MPI process could not be computed since partition size per MPI process was restricted to 10.5 million spectra.

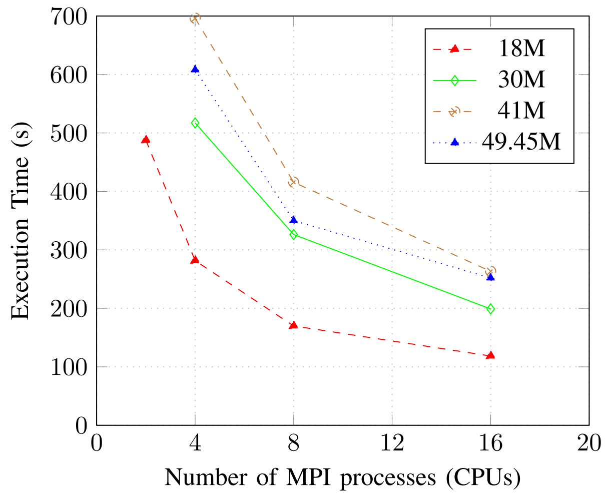

Fig. 9. This plots shows the execution time results (cyclic policy) with increasing number of MPI processes (CPUs) with increasing index size.

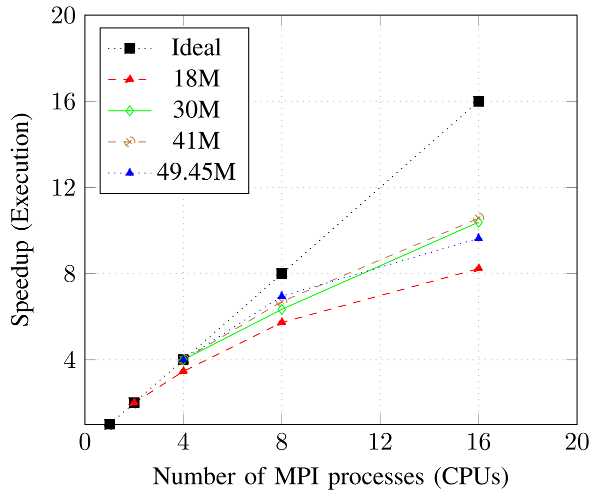

Fig. 10. The speedup results for execution time (cyclic policy) are bounded by Amdahl's law. The scalability improves as in the index size increases since the query time portion increases in total execution time.

VI. Discussion

The memory footprint results show that the distributed SLM-Index does not impose any significant memory footprint overhead which allows the distributed SLM-Index to maintain its $\sim 3.5\times$ memory footprint advantage over MSFragger [5] index, as also discussed in [6]. The temporary footprint incurred while index construction can be eliminated by enabling partitioning at each MPI process. The load imbalance results have significant importance in determining the system efficiency since it indicates the time when one or more compute resources are (idling) waiting for the stalling compute nodes to finish their task. Assume that there are N CPUs in the distributed system, with Load Imbalance $LI = x$ over the balanced system compute time T_{avg}, then the approximate total CPU time T_{wst} wasted by the system is given as:

$$T_{wst} = xNT_{avg}$$

Using Eq. 1, we have:

$$T_{wst} = N\Delta T_{max}$$

This implies that for a system with 16 CPUs $N = 16$, affected by $\Delta T_{max} = 80s$ over $T_{avg} = 100s$ i.e. $x = 0.8$ will result in wasted CPU time $T_{wst} = 1280s$ or $12.8\times$ performance degradation. This implies that even a small percentage of load imbalance can be amplified to huge performance degradation if there are large number of machines in the system. Note that in this example, the apparent wall clock time based performance degradation would appear to be $80s$ or $0.8\times$ which is not true in parallel systems. Fig. 11 shows the CPU time speedup results achieved by LBE based data partitioning for experiments discussed in Section V-C over conventional (chunk) based partitioning.

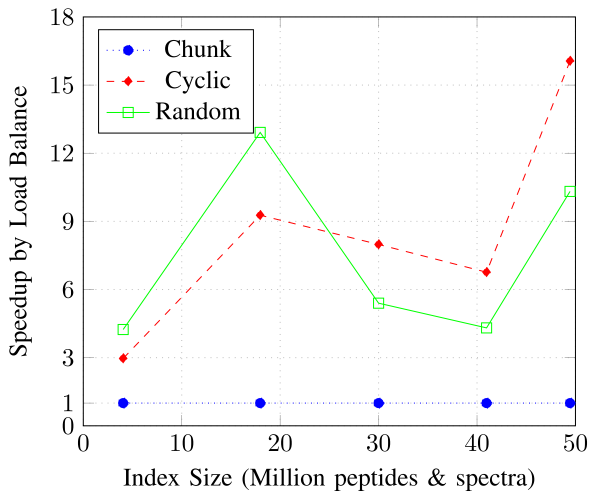

Fig. 11. The LBE cyclic and random partitioning enable orders of magnitude speedup (average: $\sim 8.6\times$ and $\sim 7.5\times$ respectively) over conventional chunk based data partitioning with 16 CPUs (16 partitions).

VII. Conclusion

In this paper, we presented a computational load balancing algorithm, called LBE, for efficient distribution of theoretical spectra data across a distributed system to allow distributed *shared-peak* based data filtration. The LBE algorithm clusters the similar spectra into groups. The spectra groups are then distributed among system machines to achieve a near similar data distribution across the system. The results show that the load imbalance across the system is $\leq 20\%$ when the data is partitioned using LBE based data distribution method. Balancing the load across the system allows a speedup of order of magnitudes over conventional partitioning method. The results also show that there is no significant memory footprint overhead imposed by distributed implementation over SLM-Index shared memory implementation. Further, the scalability results show a near ideal linear speedup for query time results. Peptide identification by comparing MS/MS spectral data obtained from a mass-spectrometer against a database of reference spectra is an extremely compute and memory intensive task. The scalability and performance of existing shared memory system based peptide identification algorithms is mostly limited by the computation, memory and I/O bandwidth bottlenecks. We propose employing distributed systems to alleviate these bottlenecks. Finally, since the parallel and distributed peptide search algorithms are in early stages of development, this work will horizon several future research problems that are pertinent to mitigate the difference between the rate of MS/MS data generation and identification.

VIII. Future Work

Our preliminary results have shown that the existing *shared-peak* count based query algorithms are bounded by the I/O bandwidth for many parallel cores. On the other hand, since the multicore systems are symmetrical and shared memory is accessible, the load in shared memory system can be efficiently balanced. Therefore, to exploit both machine and core level parallelism, we are investigating a hybrid OpenMP and MPI based approach to maximize the system throughput. We are also working towards a load-predicting model for heterogeneous memory-distributed architectures. We expect to report excellent scalability results on large supercomputing machines for our load-balancing strategy.

Funding

This research was supported by National Institute of General Medical Sciences (NIGMS) of the National Institutes of Health (NIH), United States under Award Number R15GM120820, and National Science Foundations (NSF) under Award Numbers NSF CRII CCF-1855441, and NSF CAREER ACI-1651724. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or that of National Science Foundation.

References

[1] A. I. Nesvizhskii, F. F. Roos, J. Grossmann, M. Vogelzang, J. S. Eddes, W. Gruissem, S. Baginsky, and R. Aebersold, "Dynamic spectrum quality assessment and iterative computational analysis of shotgun proteomic data toward more efficient identification of post-translational modifications, sequence polymorphisms, and novel peptides," *Molecular & Cellular Proteomics*, vol. 5, no. 4, pp. 652–670, 2006.

[2] R. Aebersold and M. Mann, "Mass spectrometry-based proteomics," *Nature*, vol. 422, no. 6928, p. 198, 2003.

[3] J. K. Eng, B. C. Searle, K. R. Clauser, and D. L. Tabb, "A face in the crowd: recognizing peptides through database search," *Molecular & Cellular Proteomics*, pp. mcp–R111, 2011.

[4] F. M. White, "The potential cost of high-throughput proteomics," *Sci. Signal.*, vol. 4, no. 160, pp. pe8–pe8, 2011.

[5] A. T. Kong, F. V. Leprevost, D. M. Avtonomov, D. Mellacheruvu, and A. I. Nesvizhskii, "Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry–based proteomics," *Nature methods*, vol. 14, no. 5, p. 513, 2017.

[6] M. Haseeb, M. G. Awan, A. Cadigan, and F. Saeed, "Slm-transform: A method for memory-efficient indexing of spectra for database search in lc-ms/ms proteomics," *bioRxiv*, 2019.

[7] H. Chi, C. Liu, H. Yang, W.-F. Zeng, L. Wu, W.-J. Zhou, X.-N. Niu, Y.-H. Ding, Y. Zhang, R.-M. Wang, *et al.*, "Open-pfind enables precise, comprehensive and rapid peptide identification in shotgun proteomics," *bioRxiv*, p. 285395, 2018.

[8] B. J. Diament and W. S. Noble, "Faster sequest searching for peptide identification from tandem mass spectra," *Journal of proteome research*, vol. 10, no. 9, pp. 3871–3879, 2011.

[9] W. Bittremieux, P. Meysman, W. S. Noble, and K. Laukens, "Fast open modification spectral library searching through approximate nearest neighbor indexing," *Journal of proteome research*, vol. 17, no. 10, pp. 3463–3474, 2018.

[10] O. S. Skinner and N. L. Kelleher, "Illuminating the dark matter of shotgun proteomics," *Nature biotechnology*, vol. 33, no. 7, p. 717, 2015.

[11] J. M. Chick, D. Kolippakkam, D. P. Nusinow, B. Zhai, R. Rad, E. L. Huttlin, and S. P. Gygi, "A mass-tolerant database search identifies a large proportion of unassigned spectra in shotgun proteomics as modified peptides," *Nature biotechnology*, vol. 33, no. 7, p. 743, 2015.

[12] J. Griss, Y. Perez-Riverol, S. Lewis, D. L. Tabb, J. A. Dianes, N. del Toro, M. Rurik, M. Walzer, O. Kohlbacher, H. Hermjakob, *et al.*, "Recognizing millions of consistently unidentified spectra across hundreds of shotgun proteomics datasets," *Nature methods*, vol. 13, no. 8, p. 651, 2016.

[13] S. Tanner, P. A. Pevzner, and V. Bafna, "Unrestrictive identification of post-translational modifications through peptide mass spectrometry," *Nature protocols*, vol. 1, no. 1, p. 67, 2006.

[14] D. L. Tabb, A. Saraf, and J. R. Yates, "Gutentag: high-throughput sequence tagging via an empirically derived fragmentation model," *Analytical chemistry*, vol. 75, no. 23, pp. 6415–6421, 2003.

[15] S. Tanner, H. Shu, A. Frank, L.-C. Wang, E. Zandi, M. Mumby, P. A. Pevzner, and V. Bafna, "Inspect: identification of posttranslationally modified peptides from tandem mass spectra," *Analytical chemistry*, vol. 77, no. 14, pp. 4626–4639, 2005.

[16] M. Mann and M. Wilm, "Error-tolerant identification of peptides in sequence databases by peptide sequence tags," *Analytical chemistry*, vol. 66, no. 24, pp. 4390–4399, 1994.

[17] S. Dasari, M. C. Chambers, R. J. Slebos, L. J. Zimmerman, A.-J. L. Ham, and D. L. Tabb, "Tagrecon: high-throughput mutation identification through sequence tagging," *Journal of proteome research*, vol. 9, no. 4, pp. 1716–1726, 2010.

[18] D. Beyter, M. S. Lin, Y. Yu, R. Pieper, and V. Bafna, "Proteostorm: An ultrafast metaproteomics database search framework," *Cell systems*, vol. 7, no. 4, pp. 463–467, 2018.

[19] M. David, G. Fertin, H. Rogniaux, and D. Tessier, "Specoms: a full open modification search method performing all-to-all spectra comparisons within minutes," *Journal of proteome research*, vol. 16, no. 8, pp. 3030–3038, 2017.

[20] M. Bern, Y. Cai, and D. Goldberg, "Lookup peaks: a hybrid of de novo sequencing and database search for protein identification by tandem mass spectrometry," *Analytical chemistry*, vol. 79, no. 4, pp. 1393–1400, 2007.

[21] H. Chi, K. He, B. Yang, Z. Chen, R.-X. Sun, S.-B. Fan, K. Zhang, C. Liu, Z.-F. Yuan, Q.-H. Wang, *et al.*, "pfind–alioth: A novel unrestricted database search algorithm to improve the interpretation of high-resolution ms/ms data," *Journal of proteomics*, vol. 125, pp. 89–97, 2015.

[22] Y. Li, H. Chi, L.-H. Wang, H.-P. Wang, Y. Fu, Z.-F. Yuan, S.-J. Li, Y.-S. Liu, R.-X. Sun, R. Zeng, *et al.*, "Speeding up tandem mass spectrometry based database searching by peptide and spectrum indexing," *Rapid Communications in Mass Spectrometry*, vol. 24, no. 6, pp. 807–814, 2010.

[23] R. C. Edgar, "Search and clustering orders of magnitude faster than blast," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[24] C. Mercier, F. Boyer, A. Bonin, and E. Coissac, "Sumatra and sumaclust: fast and exact comparison and clustering of sequences," in *Programs and Abstracts of the SeqBio 2013 workshop. Abstract*, pp. 27–29, Citeseer, 2013.

[25] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "Cd-hit: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[26] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006.

[27] M. Andreatta, B. Alvarez, and M. Nielsen, "Gibbscluster: unsupervised clustering and alignment of peptide sequences," *Nucleic acids research*, vol. 45, no. W1, pp. W458–W463, 2017.

[28] F. Saeed, J. D. Hoffert, and M. A. Knepper, "Cams-rs: clustering algorithm for large-scale mass spectrometry data using restricted search space and intelligent random sampling," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 11, no. 1, pp. 128–141, 2014.

[29] A. M. Frank, N. Bandeira, Z. Shen, S. Tanner, S. P. Briggs, R. D. Smith, and P. A. Pevzner, "Clustering millions of tandem mass spectra," *Journal of proteome research*, vol. 7, no. 01, pp. 113–122, 2007.

[30] I. Beer, E. Barnea, T. Ziv, and A. Admon, "Improving large-scale proteomics by clustering of mass spectrometry data," *Proteomics*, vol. 4, no. 4, pp. 950–960, 2004.

[31] D. Kessner, M. Chambers, R. Burke, D. Agus, and P. Mallick, "Proteowizard: open source software for rapid proteomics tools development," *Bioinformatics*, vol. 24, no. 21, pp. 2534–2536, 2008.

[32] M. Sturm, A. Bertsch, C. Gröpl, A. Hildebrandt, R. Hussong, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, A. Zerck, K. Reinert, *et al.*, "Openms–an open-source software framework for mass spectrometry," *BMC bioinformatics*, vol. 9, no. 1, p. 163, 2008.

[33] L. Martens, J. Vandekerckhove, and K. Gevaert, "Dbtoolkit: processing protein databases for peptide-centric proteomics," *Bioinformatics*, vol. 21, no. 17, pp. 3584–3585, 2005.