Naiad

RAHUL RAGHATATE[1,*] AND SNEHAL CHEMBURKAR[1]

[1] School of Informatics and Computing, Bloomington, IN 47408, U.S.A.
[*] Corresponding authors: rragahta@iu.edu, snehchem@iu.edu

paper-2, April 12, 2017

Naiad is a distributed system based on computational model called timely dataflow developed for execution of data-parallel, cyclic dataflow programs. It provides an in-memory distributed dataflow framework which exposes control over data partitioning and enables features like the high throughput of batch processors, the low latency of stream processors, and the ability to perform iterative and incremental computations. These features allow the efficient implementation of many dataflow patterns, from bulk and streaming computation to iterative graph processing and machine learning. This paper explains the Naiad Framework, its abstractions, and the reasoning behind it.

Keywords: Naiad, iterative, dataflow, graph, stream

https://github.com/cloudmesh/sp17-i524/blob/master/paper2/S17-IR-2026/report.pdf

1. INTRODUCTION

Abundance of distributed dataflow systems around the big data ecosystem has resulted in achieving many goals like high throughput, incremental updating, low latency stream processing, iterative computation. An iterative computation can be think of as some function being looped on its output iteratively until there are no changes between the input and the output and the function converges to a fixed point. On the other hand, in incremental processing, we start with initial input A_0 and produce some output B_0. At some later point, a change δA_1 to the original input A_0, leads to new input $A_1 = A_0 + \delta A_1$. Incremental model produces an incremental update to the output, so $\delta B_1 = F(\delta A_1)$ and $B_1 = \delta B_1 + B_0$. Next state computation in incremental model is based on only previous state (i.e. A0 and B0) [1].

Many data processing tasks require low-latency interactive access to results, iterative sub-computations, and consistent intermediate outputs so that sub-computations can be nested and composed. Most data-parallel systems support either iterative or incremental computations with the dataflow model, but not both. Frameworks like descendants of MapReduce [2, 3],stream processing systems [4, 5], materialized view-maintenance engines [6], provide efficient support for incremental input, but do not support iterative processing. On the other hand, iterative data-parallel frameworks like Datalog [7], recursive SQL databases [8] and systems adding iteration over MapReduce like model [9–12] provide a constrained programming model (e.g. stratified negation in Datalog) and none supports incremental input processing [13].

With a goal to find common low-level abstraction and design general purpose system which resolves above mentioned issues in computation workloads, Naiad, a timely dataflow system was developed at Microsoft by Derek G. Murray *et al.*.

Naiad provides a distributed platform for executing data parallel cyclic dataflow programs. It offers high throughput batch processing, low latency stream processing and the ability to perform iterative and incremental computations all in one framework [14].

In Naiad, the underlying timely dataflow computation framework provides a mechanism for iteration and incremental updates. Moreover, the high-level language support provides a structured means of expressing these computations. This framework can be utilized to build many expressive programming model on top of Naiad's low-level primitives enabling such diverse tasks as streaming data analysis, iterative machine learning, and interactive graph mining [13].

2. ARCHITECTURE

The Naiad architecture consists of two main components- (1) incremental processing of incoming updates and (2) low-latency real-time querying of the application state.

Figure 1 shows a Naiad application that supports real-time queries on continually updated data. The dashed rectangle represents iterative processing that incrementally updates as new data arrive [15].

From Figure 1, query path is clearly separated from the update path. This results in query processing separately on a slightly stale version of the current application state and the query path does not get blocked or incur delays due to the update processing. This also resolves complex situations: If queries shared the same path with updates, the queries could be accessing partially processed/incomplete/inconsistent states, which

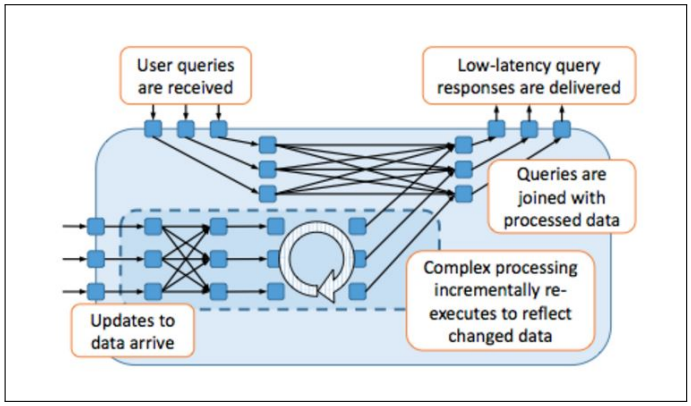

Fig. 1. Naiad Application that support real time queries on continually updated data [15].

would have to be taken care out separately.

Even though hybrid approach to assemble the application in Figure 1 by combining multiple existing systems have been widely deployed, applications built on a single platform as in Figure 1 are typically more efficient, succinct, and maintainable [15].

3. TIMELY DATAFLOW

Applications should produce consistent results, and consistency requires coordination, both across dataflow nodes and around loops [15]. Timely dataflow is a computational model that attaches virtual timestamps to events in structured cyclic dataflow graphs providing coordination mechanism that allows low-latency asynchronous message processing while efficiently tracking global progress and synchronizing only where necessary to enforce consistency. Naiad model simply checkpoints its state periodically, restoring the entire system state to the most recent checkpoint on failure. Even if it is not a sophisticated design, it was chosen in part for its low overhead. Faster common-case processing allows more computation to take place in the intervals between checkpointing, and thus, often decreases the total time to job completion [16].

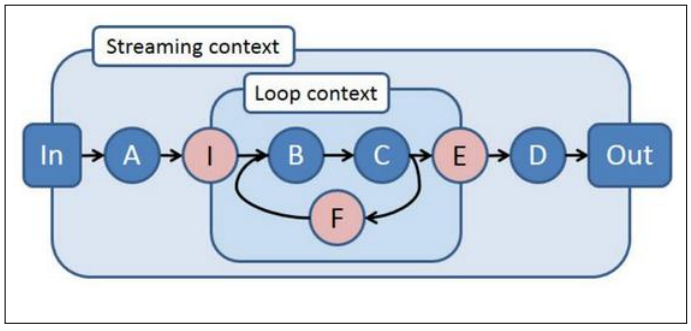

Fig. 2. A simple timely dataflow graph [15].

Consider a simple timely dataflow graph shown in Figure 2 showing a loop context nested in the top-level streaming context. Vertices A and D are not in any loop context resulting in no loop counters for their timestamps. Whereas, vertices B, C, and F are nested in a single loop context, so their timestamps have a single loop counter. The Ingress (I) and Egress (E) vertices sit on the boundary of a loop context. They monitor the loop by adding and removing loop counters to and from timestamps as messages pass through them.

Naiad being a timely dataflow computational model, utilizes similar to above mentioned computational process and provide the basis for an efficient, lightweight coordination mechanism. Timely dataflow supports the following three features:

1. Structured loops allowing feedback in the dataflow
2. Stateful dataflow vertices capable of consuming and producing records without global coordination, and
3. Notifications for vertices once they have received all records for a given round of input or loop iteration.

Structured loops and Stateful dataflow vertices allows iterative and incremental computations with low latency. Notifications makes Naiad possible to produce consistent results, at both outputs and intermediate stages of computations, in the presence of streaming or iteration [15]. The timely dataflow in Naiad is achieved by optimization in services like asynchronous messaging, iterative dataflow, progress tracking and consistency improvisation.

3.1. Asynchronous messaging

All dataflow models require some communication means for message passing between node over outgoing edges. In a timely dataflow system, each node implements an *OnRecv* event handler that the system can call when a message arrives on an incoming edge, and the system provides a *Sendmethod* that a node can invoke from any of its event handlers to send a message on an outgoing edge [16]. Messages are delivered asynchronously.

3.2. Consistency

Computations like reduction functions *Count* or *Average* include subroutines that must accumulate all of their input before generating an output. At the same time, distributed applications commonly split input into small asynchronous messages to reduce latency and buffering. For timely dataflow to support incremental computations on unbounded streams of input as well as iteration, it has a mechanism to signal when a node (or data-parallel set of nodes) has seen a consistent subset of the input for which to produce a result [16].

3.3. Iterative Graph Dataflow

A Naiad dataflow graph is acyclic apart from structurally nested cycles that correspond to loops in the program. The logical timestamp associated with each event represents the batch of input that the event is associated with, and each subsequent integer gives the iteration count of any (nested) loops that contain the node. Every path around a cycle includes a special node that increments the innermost coordinate of the timestamp. The system enforced rule restricts event handler from sending a message with a time earlier than the timestamp for the event it is handling ensuring a *partial order* on all of the pending events (undelivered messages and notifications) in the system, thus enabling efficient progress tracking [15].

3.4. Progress Tracking

The ability to deliver notifications (an event that fires when all messages at or before a particular logical timestamp have been delivered to a particular node) promptly and safely is critical. This provides timely dataflow system an ability to support low-latency incremental and iterative computation with consistent

results. Naiad can implement progress trackers to establish the guarantee that no more messages with a particular timestamp can be sent to a node [15].

4. ACHIEVING TIMELY DATAFLOW

Each event has a timestamp and a location (either a vertex or edge), and are referred as pointstamp. The timely dataflow graph structure ensures that for any locations l_1 and l_2 connected by two paths with different summaries, one of the path summaries always yields adjusted timestamps earlier than the other. For each pair l_1 and l_2, we find the minimal path summary over all paths from l_1 to l_2 using a straightforward graph propagation algorithm, and record it as $\Psi[l_1, l_2]$. To efficiently evaluate the possible relation for two pointstamps (t_1, l_1) and (t_2, l_2), we test whether $\Psi[l_1, l_2](t_1) \leq t_2$ [15].

Informally, Timely Dataflow supports directed dataflow graphs with structured cycles which is analogous to structured loops in a standard imperative programming language. This structure provides information about where records might possibly flow in the computation, allowing an implementation like Naiad to efficiently track and inform dataflow vertexes about the possibility of additional records arriving at given streaming epochs or iterations [17].

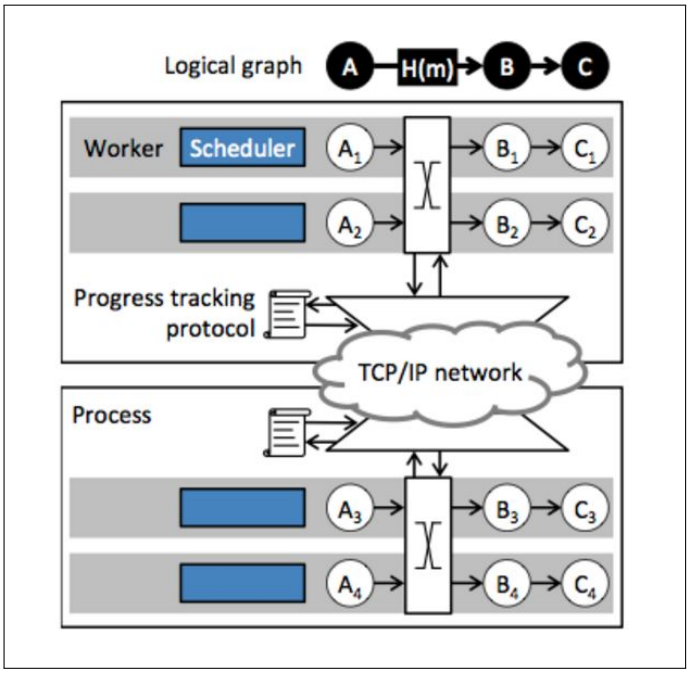

Fig. 3. The mapping of a logical dataflow graph onto the distributed Naiad system architecture [15].

5. SYSTEM IMPLEMENTATION

Naiad is high-performance distributed implementation of timely dataflow and is written in C#, and runs on Windows, Linux, and Mac OS. Figure 3 shows the schematic architecture of a Naiad cluster consisting a group of processes hosting workers that manage a partition of the timely dataflow vertices [16]. Each worker may host several stages of the dataflow graph. The workers are data nodes and keep a portion of the input data (usually a large-scale input graph, such as Twitter follower graph) in memory. So it makes sense to move computation (dataflow graph stages) to the data (the partitioned input graph)[18]. Each process participates in a distributed progress tracking protocol, in order to coordinate the delivery of notifications. All the features of C#, including classes, structs, and lambda functions, to build a timely dataflow graph from a system-provided library of generic Stream objects can be implemented. Naiad model uses deferred execution method: like adding a node to internal data flow graph at runtime while executing a method like Max on a Stream [15].

Naiad's distributed implementation also exhibits features like distributed progress tracking, a simple but extensible implementation of fault tolerance, high availability, avoidance of micro-straggler (events like packet loss, contention on concurrent data structures, and garbage collection which leads to delays ranging from tens of milliseconds to tens of seconds) which is main obstacle to scalability for low-latency workloads [15].

6. PERFORMANCE EVALUATION

Naiad's performance over supporting high-throughput, low latency, data-parallelism, batch processing, iterative graph data processing have been examined using several notable microbenchmarks by Murray *et al.* [15] such as Throughput, Latency, Protocol Optimizations, Scaling.

The hardware configuration consists of two racks of 32 computers, each with two quad-core 2.1 GHz AMD Opteron processors, 16 GB of memory, and an Nvidia NForce Gigabit Ethernet NIC. Also rack switches have 40 Gbps uplink to the core switch. Average across five trials are plotted, with error bars showing minimum and maximum values [15].

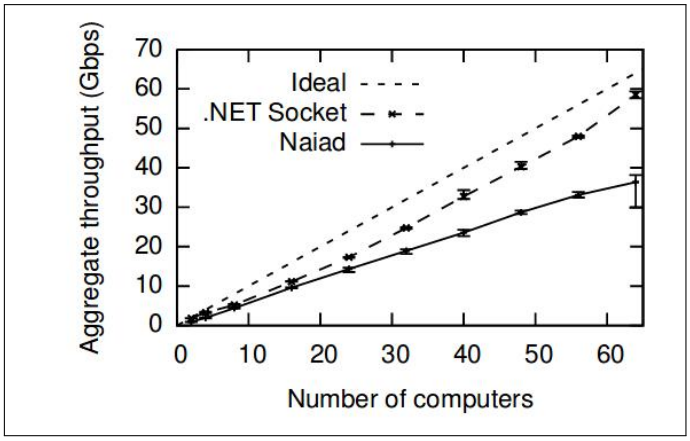

Fig. 4. Evaluation of Naiad system throughput rate on synthetic dataset [15].

6.1. Throughput

The Naiad program constructs a cyclic dataflow that repeatedly performs the all-to-all data exchange of a fixed number of records. Figure 4 plots the aggregate throughput against the number of computers with uppermost line as Ideal throughput, middle one depicting achievable throughput given network topology, TCP overheads, and .NET API costs. The final line shows the throughput that Naiad achieves for a large number of 8-byte records (50M per computer) exchanges between all processes in the cluster.

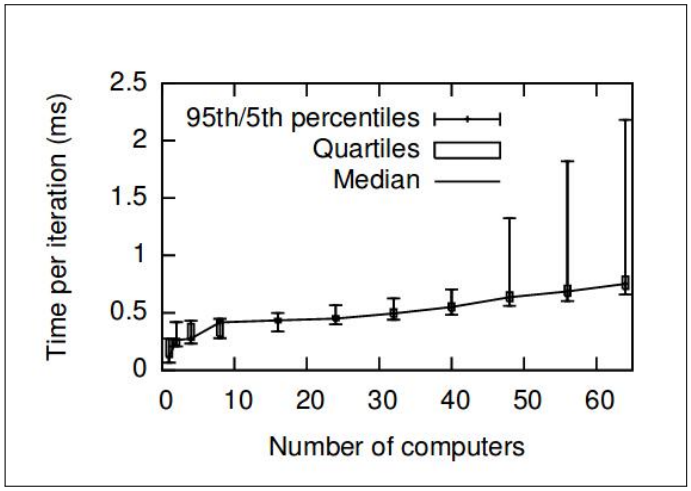

Fig. 5. Evaluation of Naiad system global barrier latency for synthetic dataset [15].

6.2. Latency

Latency microbenchmark evaluates the minimal time required for global coordination. Figure 5 plots the distribution of times for 100K iterations using median, quartiles, and 95^{th} percentile values indicating low median time per iteration and adverse impact at 95^{th} percentile mark due to increased number of computers.

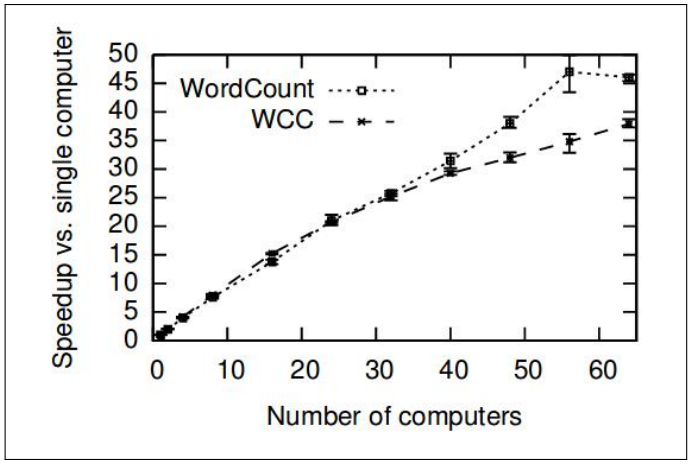

Fig. 6. Strong Scaling microbenchmark evaluation for Naiad system over synthetic dataset [15].

6.3. Scaling

Based on input size and resource availability, strong and weak scaling microbenchmarks are achieved. Keeping input fixed and increased compute resources, running time as shown in Figure 6 has been plotted for two applications, WordCount and Weakly connected components(WCC) . For WordCount, dataset used is Twitter corpus of size 128 GB uncompressed and for WCC, a random graph of 200M edges. Weak Scaling evaluation measured the effect of increasing both the number of computers and the size of the input.

Figure 7 shows the performance of WCC on random input graph with constant number of edges (18.2M) and nodes (9.1M) per computer. The running time degrades significantly to 29.4s

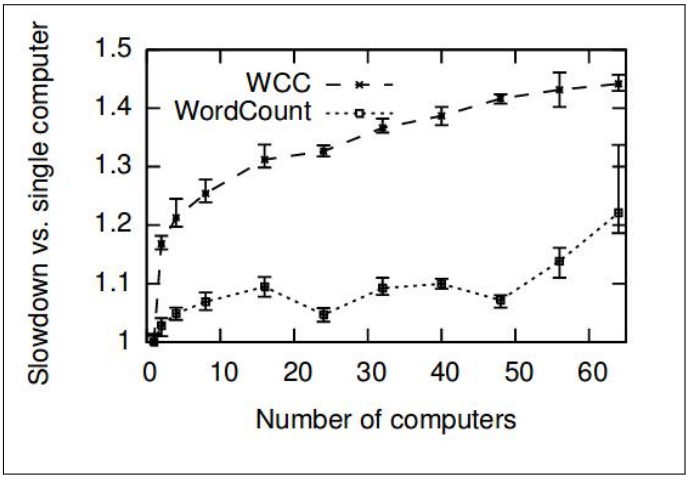

Fig. 7. Weak Scaling microbenchmark evaluation for Naiad system over synthetic dataset [15].

compared to 20.4s for 1.1B edge graph run on 64 computers cluster, relative to execution in single computer. Also the WordCount application, with 2 GB compressed input per computer, doesn't achieved perfect weak scaling, but compared to WCC, it improved.

7. USE CASES

Naiad framework has been majorly deployed for batch, streaming and graph computations serving interactive queries against the results where Naiad responds to updates and queries with sub-second latencies [15].

7.1. Batch iterative graph computation

Implementation of graph algorithms such as PageRank, strongly connected components (SCC), weakly connected components(WCC), and approximate shortest path(ASP) in Naiad requires less code complexity and provides dominant difference in running times compared to PDW [19], DryadLINQ [20],SHS [21] for Category A web graph [15, 22, 23].

7.2. Batch iterative machine learning

Naiad provides competitive platform for custom implementation of distributed machine learning. Also it is straightforward to build communication libraries for existing applications using Naiad's API. For E.g., modified version of Vowpal Wabbit (VW), an open-source distributed machine learning library which performs iterative linear regression [24]. AllReduce (processes jointly performing global averaging) implementation requires 300 lines of code, around half as many as VW's AllReduce, and the Naiad code is at a much higher level, abstracting the network sockets and threads being used [15].

7.3. Streaming acyclic computation

Latency reduction while computing the k-exposure metric for identifying controversial topics on Twitter using Kineograph, which takes snapshots of continuously ingesting graph data for data parallel computations [25].

7.4. Streaming iterative graph analytics

Twitter Analysis: To compute the most popular hashtag in each connected component of the graph of users mentioning other

users, and provide interactive access to the top hashtag in a user's connected component.

8. USEFUL RESOURCES

Naiad: A Timely Dataflow System [15], provides extensive study of Naiad's architecture, methodology of working, its distributed implementation, iterative and incremental data processing, data analysis applications, comparison with other graph processing frameworks. Moreover, [26], [27], [13] are also good resources to learn about Naiad and programming in Naiad Framework.

9. CONCLUSION

Naiad enrich dataflow computations with timestamps that represent logical points in the computational process and provide the basis for an efficient, lightweight coordination mechanism. All the above capabilities in one package allows development of High-level programming models on Naiad which can perform tasks as streaming data analysis, iterative machine learning, and interactive graph mining. Moreover, public reusable low-level programming abstractions of Naiad allow it to outperform many other data parallel systems that enforce a single high-level programming model.

ACKNOWLEDGEMENTS

This work was done as part of the course "I524: Big Data and Open Source Software Projects" at Indiana University during Spring 2017. Many thanks to Professor Gregor von Laszewski and Prof. Geoffrey Fox at Indiana University Bloomington for their academic as well as professional guidance. We would also like to thank Associate Instructors for their help and support during the course.

REFERENCES

[1] Aleksey Charapko, "One page summary: Incremental, iterative processing with timely dataflow," Blog, Feb. 2017, accessed: 09-Apr-2017. [Online]. Available: http://charap.co/one-page-summary-incremental-iterative-processing-with-timely-dataflow/

[2] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: Mapreduce for incremental computations," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 7:1–7:14, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038923

[3] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, "Nectar: Automatic management of data and computation in datacenters," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 75–88, accessed: 2017-3-22. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924949

[4] R. S. Barga, J. Goldstein, M. H. Ali, and M. Hong, "Consistent streaming through time: A vision for event stream processing," *CoRR*, vol. abs/cs/0612115, 2006, accessed: 2017-3-22. [Online]. Available: http://arxiv.org/abs/cs/0612115

[5] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, "Spade: The system s declarative stream processing engine," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1123–1134, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376729

[6] A. Gupta and I. S. Mumick, "Materialized views," A. Gupta and I. S. Mumick, Eds. Cambridge, MA, USA: MIT Press, 1999, ch. Maintenance of Materialized Views: Problems, Techniques, and Applications, pp. 145–157, accessed: 2017-3-24. [Online]. Available: http://dl.acm.org/citation.cfm?id=310709.310737

[7] S. Ceri, G. Gottlob, and L. Tanca, "What you always wanted to know about datalog (and never dared to ask)," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 146–166, Mar 1989, accessed: 2017-3-24.

[8] A. Eisenberg and J. Melton, "Sql: 1999, formerly known as sql3," *SIGMOD Rec.*, vol. 28, no. 1, pp. 131–138, Mar. 1999, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/309844.310075

[9] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 285–296, Sep. 2010, accessed: 2017-3-22. [Online]. Available: http://dx.doi.org/10.14778/1920841.1920881

[10] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: A runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 810–818, accessed: 2017-3-23. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851593

[11] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand, "Ciel: A universal execution engine for distributed data-flow computing," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 113–126, accessed: 2017-3-24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972470

[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2, accessed: 2017-3-24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[13] F. McSherry, R. Isaacs, M. Isard, and D. Murray, "Composable incremental and iterative data-parallel computation with naiad," Tech. Rep. MSR-TR-2012-105, October 2012, accessed: 2017-3-24. [Online]. Available: https://www.microsoft.com/en-us/research/publication/composable-incremental-and-iterative-data-parallel-computation-with-naiad/

[14] HU2LA, "Review of "naiad: A timely dataflow system" < huula . webdesign + ai," Blog, Sep. 2015, accessed: 23-Mar-2017. [Online]. Available: https://huu.la/blog/review-of-naiad-a-timely-dataflow-system

[15] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi, "Naiad: A timely dataflow system," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 439–455, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522738

[16] D. G. Murray, F. McSherry, M. Isard, R. Isaacs, P. Barham, and M. Abadi, "Incremental, iterative data processing with timely dataflow," *Commun. ACM*, vol. 59, no. 10, pp. 75–83, Sep. 2016, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/2983551

[17] Microsoft, "Naiad-microsoft," Web Page, Microsoft, 2017, accessed: 2017-3-22. [Online]. Available: https://www.microsoft.com/en-us/research/project/naiad/

[18] Edgar, "Metadata and the naiad post – one other good blog on data science/engineering," Blog, Jan. 2017, accessed: 23-Mar-2017. [Online]. Available: https://theinformationageblog.wordpress.com/2017/01/09/metadata-and-the-naiad-post-one-other-good-blog-on-data-scienceengineering/

[19] Microsoft, "Microsoft analytics platform system overview |microsoft," Web Page, Microsoft, 2017, accessed: 2017-3-26. [Online]. Available: https://www.microsoft.com/en-us/sql-server/analytics-platform-system

[20] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey, "Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–14, accessed: 2017-3-24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855742

[21] M. Najork, "The scalable hyperlink store," in *Proceedings of the 20th ACM Conference on Hypertext and Hypermedia*, ser. HT '09. New York, NY, USA: ACM, 2009, pp. 89–98, accessed: 2017-3-23. [Online].

Available: http://doi.acm.org/10.1145/1557914.1557933
[22] Lemur, "The clueweb09 dataset," Web Page, Red Hat, Inc., 2017, accessed: 2017-3-24. [Online]. Available: http://lemurproject.org/clueweb09/
[23] M. Najork, D. Fetterly, A. Halverson, K. Kenthapadi, and S. Gollapudi, "Of hammers and nails: An empirical comparison of three paradigms for processing large graphs," in *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, ser. WSDM '12. New York, NY, USA: ACM, 2012, pp. 103–112, accessed: 2017-3-24. [Online]. Available: http://doi.acm.org/10.1145/2124295.2124310
[24] John Lanford, "Home . johnlangford/vowpal_wabbit wiki," Code Repository, Dec. 2016, accessed: 2017-3-24. [Online]. Available: https://github.com/JohnLangford/vowpal_wabbit/wiki
[25] R. Cheng, J. Hong, A. Kyrola, Y. Miao, X. Weng, M. Wu, F. Yang, L. Zhou, F. Zhao, and E. Chen, "Kineograph: Taking the pulse of a fast-changing and connected world," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 85–98, accessed: 2017-3-22. [Online]. Available: http://doi.acm.org/10.1145/2168836.2168846
[26] Microsoft Research, "Microsoft/naiad:the naiad system provides fast incremental and iterative computation for data-parallel workloads," Code Repository, Nov. 2014, accessed: 2017-3-24. [Online]. Available: https://github.com/MicrosoftResearch/Naiad
[27] naiadquestions@microsoft.com, "Nuget gallery|naiad - core 0.5.0-beta," Web Page, NET Foundation, Nov. 2014, accessed: 2017-3-26. [Online]. Available: https://www.nuget.org/packages/Microsoft.Research.Naiad/