Robust Distributed Orthogonalization Based on Randomized Aggregation

Wilfried N. Gansterer, Gerhard Niederbrucker, Hana Straková and Stefan Schulze Grotthoff
University of Vienna, Austria
Research Group Theory and Applications of Algorithms
wilfried.gansterer@univie.ac.at, gerhard.niederbrucker@univie.ac.at,
hana.strakova@univie.ac.at, stefan.schulzegrotthoff@gmail.com

ABSTRACT

The construction of distributed algorithms for matrix computations built on top of distributed data aggregation algorithms with randomized communication schedules is investigated. For this purpose, a new aggregation algorithm for summing or averaging distributed values, the push-flow algorithm, is developed, which achieves superior resilience properties with respect to node failures compared to existing aggregation methods. On a hypercube topology it asymptotically requires the same number of iterations as the optimal all-to-all reduction operation and it scales well with the number of nodes. Orthogonalization is studied as a prototypical matrix computation task. A new fault tolerant distributed orthogonalization method (rdmGS), which can produce accurate results even in the presence of node failures, is built on top of distributed data aggregation algorithms.

Categories and Subject Descriptors

G.1.0 [**Numerical Analysis**]: General—*Numerical algorithms, Parallel algorithms*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*Computations on matrices*

General Terms

Algorithms, Performance, Reliability

1. INTRODUCTION

Algorithms for future large-scale computer systems have to be designed such that resilience to various types of failures is provided and that as little as possible synchronization between nodes is required. The basic idea underlying this paper is to investigate the construction of suitable distributed algorithms for matrix computations built on top of *distributed data aggregation algorithms* (*DDAAs*). Such algorithms can be seen as distributed versions of all-to-all reduction operations, in particular for summing or for averaging the elements of a long vector distributed over many nodes. Consequently, distributed versions of basically all types of BLAS operations can potentially be constructed based on DDAAs.

Problem Setting

We consider a large-scale computer system with N nodes arranged in a fixed but otherwise arbitrary topology as target system for the computations. Our focus is on *distributed* algorithms for such large-scale systems, which we clearly distinguish from *parallel* algorithms. Parallel algorithms are usually designed for small to medium-sized static and reliable systems, where synchronized computation across nodes in the network can be guaranteed. However, these algorithms have major drawbacks in possibly decentralized large-scale systems with arbitrary topologies and potentially unreliable components (nodes or communication links), where synchronization of the nodes may be difficult to achieve. Distributed algorithms provide much greater flexibility with respect to the hardware infrastructure than classical parallel algorithms. They neither rely on a fixed communication schedule nor on full synchronization across the nodes. Moreover, they have the potential for producing meaningful results even in the presence of link or node failures. More generally speaking, distributed algorithms are useful in all computations over large-scale computing systems where (i) the nodes do not have complete *global* information about the system, but predominantly only local information about their neighborhood, and/or (ii) the system is not static, but may change dynamically (e. g., due to hardware failures).

We investigate distributed algorithms based on *gossiping protocols*. They are attractive in such situations, because due to their randomized information exchange they do not require static or reliable hardware infrastructure. If communication is restricted to the local neighborhood of each node, the number of iterations required tends to scale logarithmically with the number of nodes in the system, which is asymptotically the same as in optimized all-to-all reduction operations. Moreover, due to their iterative structure they can deliver results at reduced accuracy levels for a communication cost which is proportional to the target accuracy.

Related Work

Approaches for achieving fault tolerance in parallel and distributed systems have been investigated at various levels. At the MPI level, fault tolerant MPI [8] has been investigated. At the parallel application level, the standard approach is (*coordinated*) *checkpointing* followed by *rollback recovery* in case of a failure. It is unclear how to scale this method to

ScalA'11, November 14, 2011, Seattle, Washington, USA.

very large systems [26]. Several alternatives have been proposed [10, 21, 20], but they all have their drawbacks. In contrast to checkpointing, *redundant computing* [9, 22, 6] can handle multiple failures without recovery overhead.

A purely algorithmic way of achieving fault tolerance for high level matrix operations is the technique of *algorithm-based fault tolerance* (*ABFT*) [11]. The basic idea of ABFT is to extend the input matrices by checksums and to adapt the algorithms such that these checksums get updated correctly and can consequently be used for detecting and recovering from errors. Classically, ABFT was designed for handling a prescribed amount of miscalculations with a high probability [11], whereas more recently also fail-stop failures, where nodes entirely crash, were considered [4]. Beside the extension to fail-stop failures there are also efforts into ABFT methods which make use of the inherent redundancy of the data distribution across the nodes to recover from failures, which results in methods where only failure situations lead to an overhead [3].

Distributed algorithms based on *gossiping* (or *epidemic*) protocols have been discussed extensively in the literature [2, 17, 5, 1]. However, almost all existing work focuses on very simple operations, such as information dissemination [14], aggregation [12], network organization [23], or computing separable functions [18].

For our objective, DDAAs for the distributed computation of sums and averages, such as the *push-sum algorithm* [15], are most relevant. Only recently, distributed algorithms for more complex matrix computations based on DDAAs, such as the *dmGS* algorithm for distributed QR factorization [25] or a distributed orthogonal iteration method [16, 24] (both based on the push-sum algorithm) have been investigated. The potential of DDAAs for providing a high degree of resilience is mentioned in the literature at various places (see, e. g., [19]), but we are not aware of any work which investigates the challenges arising when trying to tap the full potential.

2. RESILIENCE OF DISTRIBUTED DATA AGGREGATION METHODS

For a structured discussion of the differences between the various methods we distinguish five types of *failures* in the system under consideration. Note that we order the failure types according to increasing difficulty for recovering from them at the algorithmic level.

(F1) Reported temporary unavailability of links/nodes

(F2) Unreported loss or corruption of a message

(F3) Reported permanent node failures

(F4) Unreported corruption of data (e. g., bit flip)

(F5) Unreported permanent node failures

While dealing with (F1) is generally easy for any randomized method with flexible communication schedules, the coverage of (F2)–(F5) is a lot harder since those failures usually introduce a (temporary) error from which the system has to recover properly. In the case of (F5) one has to additionally define whether the initial data of a failed node should be included in the target aggregate or not, where the former is usually harder.

In the *push-sum algorithm* [15] each node i iteratively updates a local vector $v_i := (s_i, w_i)$. The s_i are initialized with the values to be aggregated, and the w_i are weights. By consecutively sending fractions of the local vector v_i to randomly chosen neighbors (which add the received values to their local values), all local estimates s_i/w_i converge linearly either to the sum or the average of the distributed values, depending on how the weights have been initialized [15]. In [7] a more robust version of the push-sum algorithm, called *LiMoSense*, is derived by keeping a history (i. e., the sum) of sent and received values along each communication link. By always sending the full history the receiver of a message can easily tolerate missed (or wrong) values. To keep the steadily growing histories small, a bidirectional cancellation operation is proposed in [7].

Another approach is *flow updating* [13], where the nodes keep their initial values local and only share *flows* with their neighbors. For each communication link, both attached nodes maintain a *flow variable* which represents the overall balance of communicated local values along this link. For communicating local values along a link, the sender does not transmit the local values directly, but instead adds them to the corresponding flow variable and transmits the flow variable. The receiver updates its own flow variable by the negated received flow. Consequently, as in network flow algorithms, the overall flow across each link is zero (*flow conservation*) if no failures occur. This flow conservation is the key idea of this method for recovering from failures, since recovering from a failure corresponds to (re)establishing flow conservation, which is achieved after each successful communication across a link.

While the push-sum algorithm can only recover from failure type (F1), LiMoSense can recover from failure types (F1), (F2) and (F3). Flow updating can recover from failure types (F1)–(F4). The major drawback of flow updating is that no formal analysis is available in [13], whereas for the push-sum algorithm [15] and for LiMoSense [7] proofs of correctness and convergence (speed) have been given.

3. A NEW APPROACH TO DISTRIBUTED ORTHOGONALIZATION

When constructing distributed algorithms for matrix computations based on sequences of distributed data aggregation algorithms, resilience aspects have to be addressed first at the level of a single DDAA. We present the new *push-flow algorithm*, which has superior fault tolerance properties compared to existing distributed data aggregation algorithms. As a prototypical example for matrix computations based on DDAAs, we then present the new distributed orthogonalization algorithm *rdmGS* which is resilient to node failures.

We first evaluate the strengths and weaknesses of existing distributed data aggregation algorithms. Then, we present a new algorithm which overcomes drawbacks of existing methods in terms of resilience. Finally, we show how DDAAs can be used as building blocks for developing a scalable and fault tolerant distributed orthogonalization/QR factorization method as a prototypical matrix computation task.

On the lower level of distributed data aggregation algorithms our approaches do not require any checkpointing or redundant computing. On the higher level (distributed QR factorization) our approach fits into the basic concept of redundant computing (without extra hardware, though). In

contrast to checkpointing, our approaches presented in this paper do not store data in regular intervals, but they react on failures which occurred in the system and do not assume any common external storage space. In our algorithms failed nodes do not have to be repaired or replaced by spare nodes to preserve constant number of nodes.

3.1 The Push-Flow Algorithm

Despite several drawbacks of the flow updating algorithm in terms of uncompetitive scalability and convergence speed as well as lack of formal analysis in [13], the principal idea of sharing flows has several conceptual advantages over keeping histories as in LiMoSense. First, a valid flow across a link can be established from any direction, while in a history-based approach a specific direction is needed to tolerate failures like (F2). Second, in contrast to a steadily increasing history value, flow variables remain bounded by definition. Third, flows achieve higher fault tolerance, since history-based approaches are limited to transmission related failures (F1)–(F3), whereas a flow-based approach can also recover from purely local failures of a node like (F4).

Motivated by this observation, we integrated the flow concept into the push-sum algorithm by translating each transmission of mass, i. e., the sending of fractions of the local values v_i, into a transmission of a flow as in the flow updating method. In the resulting new *push-flow algorithm* each node i maintains flow vectors $f_{i,j}$ for the balance of values sent to neighbor j in its neighborhood $\mathcal{N}_i$. In push-flow the local estimates for a node i are (contrary to flow updating) computed as the initial value v_i minus the sum over all flows $f_{i,j}$.

The push-flow algorithm is equivalent to the push-sum algorithm in failure-free networks, since for identical communication patterns both algorithms produce identical local values v_i. The equivalence to push-sum also highlights that no averaging technique (which slows down convergence) like in flow-updating is needed and that the number of iterations to achieve an ϵ-approximate of the true aggregate is bounded by $\mathcal{O}(\log N + \epsilon^{-1})$. We observe this bound empirically by simulating systems with hypercube interconnects and up to $N = 2^{20}$ nodes.

Summarizing, the push-flow algorithm achieves the best resilience of all DDAAs so far and is particularly interesting in cases where the local data of permanently failed nodes can be excluded from the final aggregate and thus no redundancy in storing the original data is required. In situations where it is required that the original data of *all* nodes is aggregated, redundancy in the original data has to be introduced as discussed in the next section.

3.2 Robust Distributed Orthogonalization

The distributed orthogonalization of the columns of the matrix $A \in \mathbb{R}^{n\times m}$ over a system with N nodes is a prototypical example where the loss of original data usually resulting from a permanent node failure *cannot* be tolerated. Distributed modified Gram-Schmidt orthogonalization (*dmGS*) based on the push-sum algorithm for computing the QR decomposition $A = QR$ with $Q \in \mathbb{R}^{n\times m}$ and $R \in \mathbb{R}^{m\times m}$ has been presented in [25]. It assumes that matrices A and Q are distributed rowwise over the N nodes. If a node permanently fails during the execution of dmGS, its local part of A as well as the part of Q computed so far are permanently lost and as a consequence it becomes impossible to orthogonalize all original vectors.

We present *robust dmGS* (*rdmGS*), which increases the fault tolerance compared to dmGS and produces a complete and accurate QR factorization of a matrix A even if one or several nodes fail permanently during the computation.

The pivotal idea of rdmGS is to maintain redundant copies of all nodes' relevant local data at more than one active node at all times. Each node is responsible for several rows of matrix A and for the parts of corresponding rows of Q which have been computed so far, which we call the node's *primary* data. Moreover, each node also stores *backup* data of other nodes' primary data. At every point in time, $r-1$ backup copies of node k's primary data are stored on $r-1$ distinct other active nodes in the network which act as backup nodes for node k. If node k fails, its primary data is still available on $r-1$ other active nodes. One of these nodes takes over the primary responsibility for this data, and selects another active node (usually in its neighborhood) which replaces himself. As a result, again r copies of the data of the failed node k exist in the system. The parameter r is a measure for the resilience as well as for the overhead of rdmGS. Larger r allows for tolerating more simultaneous node failures, albeit at higher cost.

This concept operates successfully under the following assumptions: (*i*) the topology of the system stays connected despite all occurring node failures, (*ii*) a reliable and efficient mechanism is available for determining whether a node (usually in the neighborhood) is alive or not, and (*iii*) the employed distributed data aggregation algorithm has to tolerate at least failure types (F1)–(F3).

The main building block of our distributed orthogonalization method is a distributed algorithm for computing the sum of values distributed over the network. The properties of rdmGS depend strongly on the properties of the underlying DDAA. Besides computing a distributed sum by a DDAA no communication between nodes is needed since all other computation can be done locally. Since all interaction with other nodes is concentrated in the data aggregation, good scalability of the underlying DDAA also implies good scalability of rdmGS. Therefore, an important challenge in the design of a scalable distributed orthogonalization method which is resilient to node failures lies in the setup of the underlying DDAA or computing the sum of distributed values.

We developed a simulation model for rdmGS based on the ns-3 network simulator and simulated distributed orthogonalization on a network arranged in a 5-cube topology and for exponentially distributed times until failure. The simulation results showed that rdmGS can tolerate a certain extent of node failures (on average, more than one fifth of the nodes failed) and still compute accurate QR factorizations.

4. CONCLUSIONS

Distributed algorithms with randomized communication can be attractive for unreliable or unstable large-scale systems in terms of fault tolerance. We have discussed resilience of methods for distributed computation of sums or averages. Moreover, we have investigated a distributed orthogonalization method on top of distributed data aggregation algorithms which is resilient to node failures.

Acknowledgments

The research was funded by the Austrian Science Fund (FWF) in project S10608.

5. REFERENCES

[1] T. Aysal, M. Yildiz, A. Sarwate, and A. Scaglione. Broadcast gossip algorithms for consensus. *IEEE Trans. Signal Processing*, 57(7):2748 –2761, 2009.

[2] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory, Special issue of IEEE Transactions on Information Theory and IEEE ACM Transactions on Networking*, 52:2508–2530, 2006.

[3] Z. Chen. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 73–84, New York, NY, USA, 2011. ACM.

[4] Z. Chen and J. Dongarra. Algorithm-based fault tolerance for fail-stop failures. *Parallel and Distributed Systems, IEEE Transactions on*, 19(12):1628 –1641, 2008.

[5] A. Dimakis, A. Sarwate, and M. Wainwright. Geographic gossip: Efficient averaging for sensor networks. *IEEE Trans. Signal Processing*, 56(3):1205–1216, 2008.

[6] C. Engelmann, H. H. Ong, and S. L. Scott. The Case for Modular Redundancy in Large-Scale High Performance Computing Systems. In *Proceedings of the 27th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2009*, pages 189–194. ACTA Press, Calgary, AB, Canada, 2009.

[7] I. Eyal, I. Keidar, and R. Rom. LiMoSense – Live Monitoring in Dynamic Sensor Networks. In *7th Int'l Symp. on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS'11)*, 2011.

[8] G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, J. Pjesivac-Grbovic, and J. J. Dongarra. Process fault tolerance: Semantics, design and applications for high performance computing. *International Journal of High Performance Computing Applications*, 19(4):465–477, 2005.

[9] K. Ferreira, R. Riesen, R. Oldfield, J. Stearley, J. Laros, K. Pedretti, and T. Brightwell. rMPI: increasing fault resiliency in a message-passing environment. Technical report, no. SAND2011-2488.

[10] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 989 –1000, 2011.

[11] K.-H. Huang and J. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, C-33(6):518 –528, 1984.

[12] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23:219–252, 2005.

[13] P. Jesus, C. Baquero, and P. S. Almeida. Fault-tolerant aggregation by flow updating. In *Proceedings of the 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, DAIS '09, pages 73–86, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 565–, Washington, DC, USA, 2000. IEEE Computer Society.

[15] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482–491. IEEE Computer Society, 2003.

[16] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. *Journal of Computer and System Sciences*, 74(1):70 – 83, 2008.

[17] A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41:2–7, 2007.

[18] D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 113–122, New York, NY, USA, 2006. ACM.

[19] A. Olshevsky and J. N. Tsitsiklis. Convergence speed in distributed consensus and averaging. *SIAM J. Control Optim.*, 48:33–55, 2009.

[20] J. Plank. Improving the performance of coordinated checkpointers on networks of workstations using raid techniques. In *Reliable Distributed Systems, 1996. Proceedings., 15th Symposium on*, pages 76 –85, 1996.

[21] J. Plank, K. Li, and M. Puening. Diskless checkpointing. *Parallel and Distributed Systems, IEEE Transactions on*, 9(10):972 –986, 1998.

[22] B. Schroeder and G. A. Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78(1):012022+, 2007.

[23] D. Shah. Gossip algorithms. *Found. Trends Netw.*, 3:1–125, 2009.

[24] H. Strakova and W. N. Gansterer. A distributed eigensolver based on randomized communication. Technical Report RLCTA-2011-1, University of Vienna, Research Group Theory and Applications of Algorithms, 2011. (*submitted*).

[25] H. Strakova, W. N. Gansterer, and T. Zemen. Distributed QR factorization based on randomized algorithms. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics*, 2011. *to appear*.

[26] M. Varela, K. Ferreira, and R. Riesen. Fault-tolerance for exascale systems. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, pages 1–4, 2010.