J. Parallel Distrib. Comput. 78 (2015) 53–64

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Flow updating: Fault-tolerant aggregation for dynamic networks

Paulo Jesus *, Carlos Baquero *, Paulo Sérgio Almeida *

HASLab, INESC TEC and Universidade do Minho, Portugal

HIGHLIGHTS

- We describe a fault-tolerant data aggregation algorithm for dynamic networks.
- Experimental results show it outperforms previous averaging techniques.
- It self-adapts to churn and input value changes.
- It supports node crashes and high levels of message loss.
- It works in asynchronous settings.

ARTICLE INFO

Article history:
Received 14 October 2014
Received in revised form
31 January 2015
Accepted 16 February 2015
Available online 24 February 2015

Keywords:
Distributed algorithms
Data aggregation
In-network aggregation
Fault-tolerance
Dynamic networks

ABSTRACT

Data aggregation is a fundamental building block of modern distributed systems. Averaging based approaches, commonly designated gossip-based, are an important class of aggregation algorithms as they allow all nodes to produce a result, converge to any required accuracy, and work independently from the network topology. However, existing approaches exhibit many dependability issues when used in faulty and dynamic environments. This paper describes and evaluates a fault tolerant distributed aggregation technique, Flow Updating, which overcomes the problems in previous averaging approaches and is able to operate on faulty dynamic networks. Experimental results show that this novel approach outperforms previous averaging algorithms; it self-adapts to churn and input value changes without requiring any periodic restart, supporting node crashes and high levels of message loss, and works in asynchronous networks. Realistic concerns have been taken into account in evaluating Flow Updating, like the use of unreliable failure detectors and asynchrony, targeting its application to realistic environments.

1. Introduction

With the advent of multi-hop ad-hoc networks, sensor networks and large-scale overlay networks, there is a demand for tools that can abstract meaningful system properties from given assemblies of nodes. In such settings, aggregation plays an essential role in the design of distributed applications [32], allowing the determination of network-wide properties like network size, total storage capacity, average load, and majorities. Although apparently simple, in practice aggregation has revealed itself to be a non-trivial problem in distributed settings, where no single element holds a global view of the whole system.

In the recent years, several algorithms have addressed the problem with diverse approaches, exhibiting different characteristics in terms of accuracy, time and communication trade-offs. A useful class of aggregation algorithms is based on *averaging* techniques. Such algorithms start from a set of input values spread across the network nodes, and iteratively average their values with neighbors. Eventually all nodes will converge to the same value and can estimate some useful metric.

Averaging techniques allow the derivation of different aggregation functions besides average (like counting and summing), according to the initial combinations of input values. For example, if one node starts with input 1 and all other nodes with input 0, eventually all nodes will end up with the same average $1/n$ and the network size n can be directly estimated by all of them [14].

Distributed data aggregation becomes particularly difficult to achieve when faults are taken into account (i.e., message loss and node crashes), and especially if dynamic settings are considered (nodes arriving/leaving). Few have approached the problem under these settings [27,25,11,24,15,23], proving to be hard to efficiently obtain accurate and reliable aggregation results in faulty and dynamic environments.

This paper extends the previous work on *Flow Updating* [16,19], a novel averaging approach, by presenting asynchronous versions

* Corresponding authors.
E-mail addresses: pcoj@di.uminho.pt (P. Jesus), cbm@di.uminho.pt (C. Baquero), psa@di.uminho.pt (P.S. Almeida).

http://dx.doi.org/10.1016/j.jpdc.2015.02.003
0743-7315/

of the algorithm and providing extensive evaluation results considering practical concerns such as: dynamic input value changes, realistic failure detectors and asynchronous execution with message loss. The evaluation shows that: it outperforms classic averaging-based aggregation algorithms; it is fault-tolerant (to both message loss and node crashes); it is able to efficiently support network dynamism (churn); it can be used with realistic failure detectors (and shows how these should be tuned); it can continuously aggregate under changes of input values with no need for a restart; it can be used in asynchronous settings, with variable transmission latency (and shows how timeouts can be chosen for a typical latency distribution, in a practical implementation).

The remainder of this paper is organized as follows. We briefly refer to the related work on aggregation algorithms in Section 2. Section 3 describes *Flow Updating*, a robust distributed aggregation algorithm able to work in dynamic networks. In Section 4, we evaluate the proposed approach. Finally, we make some concluding remarks in Section 5.

2. Related work

Classic approaches, like TAG [27], perform a tree-based aggregation where partial aggregates are successively computed from child nodes to their parents until the root of the aggregation tree is reached (requiring the existence of a specific routing topology). This kind of aggregation technique is often applied in practice to Wireless Sensor Networks (WSN) [28]. Other tree-based aggregation approaches can be found in [25,5,33]. We should point out that, although being energy-efficient, the reliability of these approaches may be strongly affected by the inherent presence of single-points of failure in the aggregation structure. Moreover, in order to operate on dynamic settings, a tree maintenance protocol is required to handle node arrival/departure, which may lead to temporary disconnection during the parent switching process.

Alternative aggregation algorithms based on the application of probabilistic methods can also be found in the literature. This is the case of Extrema Propagation [4] and COMP [31], which reduce the computation of an aggregation function to the determination of the minimum/maximum of a collection of random numbers. These techniques tend to emphasize speed, being less accurate than averaging approaches.

Specialized probabilistic algorithms can also be used to compute specific aggregation functions, such as COUNT (e.g., to determine the network size). This type of algorithm essentially relies on the results from a sampling process to produce an approximate estimate of the aggregate, using properties of random walks, capture–recapture methods and other statistic tools [30,11,29,24]. These approaches can provide some flexibility in dynamic settings, but are not accurate. The estimation error, present even in fault-free settings, depends on the quality of the collected sample, and the used estimator. Moreover, a sample is made available at a single node, and it can take several rounds to collect one sample. For example, the estimation error can reach 20% in Sample & Collide [30,11], and a single sampling step takes $\bar{d}T$ (where $\bar{d}$ is the average connection degree and T is a timer value that must be sufficiently large to provide a good sample quality) and must be repeated until l new samples have been observed.

The averaging approach to distributed aggregation is based on an iterative averaging process between small sets of nodes [22,15,14,7,35]. Eventually, all nodes will converge to the correct value by performing the averaging process across all the networks. These approaches are independent from the network routing topology, are often based on a gossip (or epidemic) communication scheme, and are able to produce an estimate of the resulting aggregate at every network node. Averaging techniques are considered to be robust and accurate (converge over time) when compared to other aggregation techniques, but in practice they exhibit relevant problems that have been overlooked, not supporting message loss nor node crashes (see [17] for more details). Moreover, most existing approaches rely on inefficient strategies to handle network dynamism, like the use of a restart mechanism that looses all progress.

A technique which combines the basic idea from Flow Updating with mass distribution is the MDFU algorithm, presented in [1]. This one keeps a pair of incoming–outgoing flow-like values, but which increase unboundedly. It inherits the convergence properties of the underlying mass distribution, while also being fault-tolerant and allowing input value changes. Another algorithm which keeps a pair of incoming–outgoing values that summarize past messages is the more recent Limosense [10], which adapts the classic Push-Sum [22], inheriting its convergence properties, while being also fault-tolerant and allowing input value changes and network dynamism. Recently, an algorithm named Push-Flow that combines Flow Updating with Push-Sum was described in [12].

A comprehensive survey about distributed data aggregation algorithms is found in [20].

3. Flow updating

Flow Updating [16,19] is a recent averaging based aggregation approach, which works for any network topology and tolerates faults. Like existing gossip-based approaches, it averages values iteratively during the aggregation process towards converging to the global network average. But unlike them, it is based on the concept of *flow*, providing unique fault-tolerant characteristics by performing idempotent updates.

The key idea in Flow Updating is to use the *flow* concept from graph theory (which serves as an abstraction for many things like water flow or electric current), and instead of storing in each node the current estimate in a variable, compute it from the input value and the contribution of the flows along edges to the neighbors:

$$e_i = v_i - \sum_{j \in n_i} f_{ij}. \tag{1}$$

This can be read as: the current estimate e_i in a node i is the input value v_i less the flows f_{ij} from the node to each neighbor j. The algorithm aims to enforce and explore the skew symmetry property of the flow along an edge, i.e., $f_{ij} = -f_{ji}$.

The essence of the algorithm is: each node i stores the flow f_{ij} to each neighbor j; node i sends flow f_{ij} to j in a message; a node j receiving f_{ij} updates its own f_{ji} with $-f_{ij}$. Messages simply update flows, being idempotent; the value in a subsequent message overwrites the previous one, it does not add to the previous value. If the skew symmetry of flows holds, the sum of the estimates for all nodes (the global mass) will remain constant:

$$\sum_{i \in V} e_i = \sum_{i \in V} \left(v_i - \sum_{j \in n_i} f_{ij} \right) = \sum_{i \in V} v_i. \tag{2}$$

The intuition is that if a message is lost the skew symmetry is temporarily broken, but as long as a subsequent message arrives, it re-establishes the symmetry. The reality is somewhat more complex: due to concurrent execution, messages between two nodes along a link may cross each other and both nodes may update their flows concurrently; therefore, the symmetry may not hold, but what happens is that $f_{ij} + f_{ji}$ converges to 0, and the global mass converges to the sum of the input values of all nodes. Message loss only delays convergence; it does not impact the convergence direction towards the correct value.

Enforcing the skew symmetry of flows along edges through idempotent messages is what confers Flow Updating its unique