A Scalable Architecture for Adaptive and Distributed Monitoring

By Jason Porter
Introduction

• Today systems are often large, complexed and highly distributed.

• Managing such systems with human operators prove difficult especially as the size of the systems continue to grow.

• Hence there is a need for these systems to be self-adaptive and “manage” themselves at runtime.

• Currently a lot of the work on self-adaptive systems focus on centralized control [1].

• In a large scale distributed system a centralized approach proves infeasible due to issues of scalability as well as the controller being a single point of failure.

Introduction (Cont’d)

• To address the aforementioned issues decentralized controllers are needed.
• This leads to the notion of a distributed MAPE-K.
• Each controller manages its local subsystem and “organizes” with neighboring controllers to achieve a global system state.
• This creates the problem of how to coordinate among multiple MAPE-K loops to achieve adaptability.
• This paper aims to tackle the sub-problem of coordinating among multiple “M” components.
Example of Distributed Monitoring Pattern


• The common model underlying gossip-based solutions is that of a (possibly dynamically changing) set of processes, called peers, each of which regularly exchanges information with other peers.

• The crucial aspects of the framework includes:
  • Peer selection: Gossiping protocols differ in their peer selection process. The mechanism used in the paper is a peer sampling service which allows a peer to uniformly randomly select another available peer.
  • Data exchanged: This is often application-dependent and depends on the particular application domain.
  • Data processing: This part of the framework describes how a peer deals with the received information. This is also highly application-dependent.
Gossip Application Domains

• Information Dissemination
  • Peer selection: each peer P periodically chooses $f \geq 1$ peers $Q_1, \ldots, Q_f$ uniformly at random from the entire set of currently available peers.
  • Data exchanged: a message is selected from the local cache and copied from one peer to another. In a "push" model, P forwards a message to each $Q_i$; in a pull model each $Q_i$ sends a message to P. A combination of both is also possible.
  • Data processing: effectively, nothing special is done except storing the received message for a next iteration, or passing it to a higher layer.

• An important observation of gossip-based dissemination is that the data gets spread exponentially fast through the network.
Gossip Application Domains (Cont’d)

• Peer Sampling
  • Peer selection: each peer P chooses periodically a gossip target Q from its current set of neighbors.
  • Data exchanged: the data exchanged between peers is simply a list of peers.
  • Data processing: Upon receipt of the list, the receiving peer merges the list of peers with its own list to compose a new list of neighbors. Some peers may need to be dropped from the new list due to size limitations.

• Topology Construction
  • Peer selection: the set of peers is ranked according to a given ranking function and the gossip target is chosen at random from among the first half of peers from the local cache.
  • Data exchanged: the data exchanged between peers are lists of peers.
  • Data processing: upon receipt of the message, the receiving peer merges the received list with its own, ranks the elements according to the given ranking function, and keeps the first elements (up to the sized required).
Gossip Application Domains (Cont’d)

• Resource Management
  • Peer selection: each peer P chooses periodically a gossip target Q from its current set of neighbors.
  • Data exchanged: the data exchanged between peers is status information on other peers (e.g., last reported alive message).
  • Data processing: upon receipt of the message, the receiving peer merges the received information with its own status information on nodes, effectively updating its view of other nodes.

• Data Aggregation
  • Peer selection: each node periodically chooses one other peer uniformly at random from the entire set of currently available (i.e., alive) peers.
  • Data exchanged: an application-specific data element is copied from one peer to another.
  • Data processing: a new data value is computed from the exchanged information, and which will then be used in a next gossip exchange.

- Ganglia is a scalable distributed monitoring system designed for large-scale clusters and grids consisting of federations of clusters.
- It is based on a hierarchical design and relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state.
- Each node monitors its local resources and sends multicast packets containing monitoring data on a well-known multicast address whenever significant updates occur.
- All nodes listen for metrics on the well-known multicast address and collect and maintain monitoring data for all other nodes. Thus, all nodes always have an approximate view of the entire cluster’s state and this state is easily reconstructed after a crash.
Implementation

• The implementation consists mainly of two daemons; gmond and gmetad.

• The Ganglia monitoring daemon (gmond) provides monitoring on a single cluster by implementing the listen/announce protocol and responding to client requests by returning an XML representation of its monitoring data. gmond runs on every node of a cluster.

• The Ganglia Meta Daemon (gmetad), on the other hand, provides federation of multiple clusters. A tree of TCP connections between multiple gmetad daemons allows monitoring information for multiple clusters to be aggregated.
Ganglia Architecture
Aim is to overcome the shortcomings of hierarchical approaches spanning multiple clusters where the root represents a single point of failure.

As such each cluster is equipped with an independent monitor infrastructure.

Provides richer analysis of monitored data by generating multiple data streams representing metrics collected from numerous processes on each node.

There is no intra-node communication within clusters to achieve a view of the cluster state.
Implementation

• Each monitored node is equipped with an independent collection agent, whose main duty is to ensure that each resource of interest is continuously monitored.

• The collection agent receives samples from a set of probes, performs preliminary validity checks on them, updates the resource data streams and sends them in a coded form (usually, a compression) to a dedicated collector node.

• The collector node receives the filtered and coded resource data streams, performs the necessary decompression and stores them for further analysis or a real-time plot.

• In the latter case, processing stops and the user is able to see immediately the behavior of the resource data stream. In the former case, data is made available to a distributed analyzer system.
Cluster Collection Filtering and Analysis

Monitored Node

- Based on the publish/subscriber system with decentralized message brokers.
- Propose to augment the system with application-specific integrated aggregation (ASIA), which offers the ability to convey additional information, besides published messages, as required by communicating components in an application.
- Compared to the use of an external aggregation system, the integration of aggregation with the publish/subscribe system has benefits in terms of efficiency through combining network communication for aggregation with actual messages.
Implementation

• The core idea behind ASIA is inspired by aspect-oriented programming (AOP).
• Define a set of join-points that are specific to the filtering and routing of messages in decentralized publish/subscribe systems.
• These join-points can be “advised” in order to perform aggregation of relevant information by altering and augmenting the default broker code.
• This is implemented through an aspect-oriented event broker algorithm for publish/subscribe systems that supports reflection by exposing join-points.
Implementation (Cont’d)

• The algorithm is an evolution of the standard publish/subscribe broker algorithm.

• The join-points refer to points in the execution, at which we may want to customize the algorithm's behavior.

• Execution "jumps" from a given join-point to a corresponding advice.

• In ASIA, this advice is used to invoke code that effects the computation of an aggregation function within a given broker.

• A “global” aggregation result can be computed by combining the individual aggregation computation at each broker via sending updates.

• Through the use of “precision” aggregate function computation values are only sent if the value exceeds a bound.

• Presents a benchmarking methodology for decentralized monitoring mechanisms in peer-to-peer systems.

• Allows for the achievement of a fair comparison between different approaches.

• Provides the following:
  • A list of non-functional requirements that should be satisfied by all decentralized monitoring systems.
  • A set of workloads designed to address and evaluate the requirements.
  • A set of metrics to quantify the fulfillment of the aforementioned requirements.
  • An evaluation between two decentralized monitoring systems, a tree based (hierarchical) approach and a gossip based (flat) approach.
Non-functional Requirements

• Workload-Independent Quality Aspects
  • Performance (responsiveness and validity)
  • Cost (communication)
  • Fairness (uniform distribution of either performance or cost between the peers)

• Workload-Dependent Quality Aspects
  • Scalability (divided in horizontal and vertical scalability)
  • Robustness (how the system handles unpredictable events and failures)
  • Stability (how the system deals with the random behavior of the peers)
Workload

• Baseline workload (models an idealized condition)
• Scalability workload
  • Horizontal scalability (models a growing number of peers in the system)
  • Vertical scalability
    • Vertical scalability I (models an increasing number of monitored attributes whiles number of peers remains unchanged)
    • Vertical scalability II (models increasing requests for the global view of attributes)
• Stability workload (models churn in the systems)
• Robustness workload
  • Massive join (models the doubling of the number of peers)
  • Massive leave (models 50% of peers ungracefully crashing)
Metrics

• Responsiveness (lookup time for a request for the global view).
• Cost (measures the overall traffic in the system).
• Validity (measures the monitoring error for an aggregate at a peer at a specific period in time).
• Staleness (comprises (i) the time to aggregate the data, (ii) the time to disseminate the data to another peer, as well as (iii) the lookup time for a request of the global view).
Evaluation

• Baseline workload
  • Tree-based approach
    • Has poorer results for monitoring error and staleness due to hierarchical topology.
  • Gossip-based approach
    • Flat topology leads to better results for staleness and a more consistent monitoring error.
    • Introduces the most traffic.

• Horizontal workload
  • Tree-based approach
    • Growing tree results in increased staleness and monitoring error.
    • Increase in traffic resulting in decreasing fairness.
  • Gossip-based approach
    • Nearly constant results for staleness and monitoring error.
    • Increase in traffic resulting in decreasing fairness.
Evaluation (Cont’d)

• Vertical Scalability
  • Significantly impact both mechanisms where the performance of each degrades in terms of accuracy and staleness. Both also experience a similar increase in traffic.

• Stability workload
  • Reveal that both monitoring mechanisms also suffer from the impact of churn especially in terms of the accuracy. Both also experience a similar increase in traffic and a decrease in fairness.

• Massive leave workload
  • Tree-based approach
    • Can not cope with a crash in the system at all.
    • Increase in traffic (due to rebuilding overlay) and decrease in fairness.
  • Gossip-based approach
    • Is able to handle a subset of simultaneously leaving peers but to only a certain degree.
    • Increase in traffic (due to rebuilding overlay) and decrease in fairness.
Evaluation (Cont’d)

• Massive join workload
  • Shows that both systems are able to handle a high number of simultaneously
    joining peers.
  • The increasing load however does result in some reduction in fairness.

• Overall the performance of the gossip-based approach is better than the tree-based
  approach as when they both aren’t performing similarly gossip provides better results.
Open Problem

• Need a system that is both adaptable and scalable.
• Tree-based (hierarchical approach) has more shortcomings when compared to the gossip-based (flat) approach.
• Plus the root of the tree represents a single point of failure.
• IP multicasting does not scale and can only be used if the underlying infrastructure has support for it.
• Publish/subscribe system although scalable has no notion of adaptability in the event a broker node fails.
• Gossip although adaptable and scalable has a high communication overhead as it generates a lot of network traffic.
Solution

• A scalable adaptive and distributed monitoring architecture.

• Uses the gossip protocol for its distributed, scalable and adaptable characteristics but handles the network traffic overhead through the use of “neighborhoods” and precision.

• Peers in the system are grouped according to their location to each other (neighborhoods) thus reducing traffic across the system, and inter-neighborhood communication is done via an elected peer from each neighborhood.

• To reduce the network traffic within and across neighborhoods, the notion of precision is also incorporated where updates are sent only if metrics are outside a given bound.
Layered Gossip Architecture

Neighborhood A
Neighborhood B
Neighborhood C
Conclusion

• Current approaches to distributed monitoring mostly cater to system administration.

• To be applicable to distributed autonomic systems we need an approach that is decentralized, scalable and adaptable.

• Gossip protocols are self-healing, self-organizing, symmetric, scalable and simple.

• To account for the communication overhead, a layered approach has been proposed incorporating the notion of precision.
References


