Computer Sciences Department

A Scalable Failure Recovery Model for Tree-Based Overlay Networks

Dorian Arnold
Barton Miller

Technical Report #1626

January 2008

UNIVERSITY OF WISCONSIN MADISON

A Scalable Failure Recovery Model for Tree-based Overlay Networks

Dorian C. Arnold Barton P. Miller
University of Wisconsin
[darnold,bart]@cs.wisc.edu

Abstract

We present a scalable failure recovery model for data aggregations in large scale tree-based overlay networks (TBONs). A TBON is a network of hierarchically organized processes that exploits the logarithmic scaling properties of trees to provide scalable data multicast, gather, and in-network aggregation. TBONs are commonly used in debugging and performance tools, system monitoring, information management systems, stream processing, and mobile ad hoc networks.

Our recovery model leverages inherent information redundancies in TBON computations. This redundant information is gathered from non-failed processes to compensate for computation and communication state lost due to failures. This state compensation strategy is attractive because: (1) it avoids the time and resource overheads of previous reliability approaches, which rely on explicit replication; (2) recovery is rapid and only involves a small subset of the network; and (3) it applies to many useful, complex computations. In this paper, we formalize the TBON model and its fundamental properties to prove that our state compensation model properly preserves computational semantics across TBON process failures. These properties lead to an efficient implementation of state compensation, which we use to empirically validate and evaluate recovery performance. We show that state compensation can recover from failures in extremely large TBONs in milliseconds rendering practically no application service interruption.

1. Introduction

The demand for powerful HPC systems has lead to significant increases in system sizes. Today, there exists many thousand, ten thousand and even a hundred thousand processor system, BlueGene/L, and numerous initiatives worldwide for additional petaflop scale systems with as many as a million processors. At these scales, HPC systems likely will exhibit low mean times between failures increasing the demand for efficient failure recovery models.

Tree-based overlay networks (TBONs) have been shown to provide a powerful computation model for tool and application scalability. A TBON is a network of hierarchically organized processes that exploits the logarithmic scaling properties of trees to provide scalable data multicast, data gather, and in-network aggregation. As early as 1980, Ladner and Fischer observed that hierarchical decomposition in the form of parallel prefix computations can be used for efficient processing [27]. Today, TBONs are used for scalable multicast infrastructures [31], for data aggregation services [3, 6, 20, 34], in tools [35, 36, 38], information management systems [32, 41], stream processing [5], and mobile ad hoc networks (MANETs) [29, 42].

Our target domain is extremely large scale TBONs with high throughput, low latency requirements for which current reliability techniques do not provide scalable solutions. This work is guided by three key observations. First, explicit data replication for failure recovery often leads to inefficient, high-overhead mechanisms. However, a fundamental property of stateful TBON computations is that the computational structure has inherent redundancies: as information is propagated from the leaves of the tree toward the root, aggregation state, which generally encapsulates the history of processed information, is replicated at successive levels in the tree. Second, weak data consistency models are amenable to efficient implementations and do not overly restrict the types of useful tree-based computations. Other researchers have proposed weaker state consistency models, like *eventual consistency* [5, 10, 22, 28, 32] and *equivalent recovery* [22] where post-failure output is equivalent, but not identical, to the output of a non-failed execution. In each of these cases, however, the resulting recovery model still relies on explicit replication. Three, recovery models that require process coordination or global consensus are inherently non-scalable. Conversely *uncoordinated* protocols like in some checkpointing protocols [] yield good performance; however these protocols still require coordination to recover to a consistent state. Perhaps due to such recovery complexities, uncoordinated checkpointing is not common in practice. We have leveraged these observations to develop a scalable recovery model, called *state compensation*, which exploits the inherent TBON information redundancies and weak consistency models to yield the following features:

- No additional computational, network, or storage overhead during normal execution,
- Completely distributed failure recovery with no process coordination amongst the small subset of the entire tree that participates in recovery, and
- Applicability to broad categories of simple and complex TBON computations

The first sections of this paper contain a theoretical treatment of state compensation, and the latter sections describe

PPoPP '08 date, City.

the design, implementation, and evaluation of an implementation based on the MRNet [34] TBON prototype. We formalize the general TBON computational model in Section 2 and use this model to prove that state compensation properly recovers from failure, completely preserving the semantics of the original computation. It is imperative that we base such assertions on proofs rather than ad hoc reasoning to prove that our fault tolerance mechanisms indeed guarantee correctness. To this end, in Section 3, we establish two key theorems: the TBON Output Theorem, which says that the output of a TBON computation depends only upon the computational state at the root process and the in-flight data, and the All-encompassing Leaf State Theorem, which says that the state at the leaves of any sub-tree subsume the computational and communication state throughout the rest of the TBON. The formal model helps us understand the applicability and limitations of our approach. The model also provides a road map for an efficient implementation. Specifically, the combination of the two properties above results in our recovery model described in Section 4 and its implementation in Section 5. In Section 6, we empirically evaluate our state compensation strategy and show that failure recovery is both efficient and scalable. Finally, after describing related research in Section 7, we close with our conclusions.

2. The TBON Model

The formal definition of our computational model is derived from our experiences with the MRNet TBON prototype [1, 2, 34, 35, 37]. For simplicity, we often depict a completely balanced, binary tree, but the model generalizes and only requires that the tree be connected. We also discuss our model's generality by discussing a variety algorithms that are well-suited for TBONs.

2.1 Computational Specification

TBON's are used to execute scalable analyses on continuously streaming data. As shown in Figure 1, the application backends at the TBON leaves are input data producers, and the application front-end at the root is the consumer of computation output. Root, leaf, and internal processes are called *communication processes* and labeled CP_i, where i is a unique identifier. CP_0 is always the TBON root. We presume that all processes know the TBON topology via information relayed during tree (re)configuration protocols.

Communication processes stream data *packets* to each other via a reliable transport like TCP. Communication processes have enumerated input channels; input packets are described by $in_n(CP_i, j)$, which specifies is the n^{th} input to CP_i on its j^{th} channel. A *channel's state* is its incident vector of in-transit packets: $cs_{m,n}(CP_i, j)$ is the vector of in-transit packets to CP_i on its j^{th} channel when CP_i has filtered m packets from this channel and the channel's source has sent n packets, $m \leq n$:

$$cs_{m,n}(CP_i, j) = [in_{m+1}(CP_i, j), in_{m+2}(CP_i, j), \ \ldots \ , in_n(CP_i, j)] \quad (1)$$

$cs(CP_i)$ represents the union of the state of all of CP_i's input channels. If the channel between two processes is empty, then we say those two processes are *synchronized*. Finally, analogous to input packets, $out_n(CP_i)$ is CP_i's n^{th} filter output. Naturally, sets of output, one from each child, become the parent's inputs:

$$\{out_n(CP_j), out_n(CP_k)\} = \{in_n(CP_i, l), in_n(CP_i, m)\} = in_n(CP_i), \quad (2)$$

where CP_j and CP_k are sources for CP_i's l^{th} and m^{th} channels; $in_n(CP_i)$ denotes the CP_i's n^{th} set of inputs.

2.2 Data Aggregation

Packet filters perform data aggregation on application packets. Filters and packets have type attributes; a filter can only be applied to similarly typed packets. Specifically, a filter, f, is a function that takes as input a set of packets and outputs a single (potentially null-valued) packet. The general TBON model allows multiple outputs, but so far in practice we have found this unnecessary. We adopt the dataflow model where a filter executes when an input from every channel is available. In our model, filters operate on specific data streams. There can be multiple active streams each with its own filter; we do not discuss multiple streams of dataflow, but our concepts readily extend to those cases.

Filter state Stateful aggregations use persistent *filter state* to carry side-effects from one invocation to the next; $fs_n(CP_i)$ is CP_i's state after n filter invocations. Consider the *sub-graph folding* filter [35], which continuously merges sub-graphs (inputs) into a single graph (output). Each communication process stores the current merged graph encapsulating the history of sub graphs filtered by that process as persistent state. As new sub-graphs (inputs) arrive, the filter outputs any updates to its current merged graph (state). In Section 2.3, we discuss how these concepts, state as history and inputs/outputs as incremental updates, generally apply to both simple and complex stateful filters.

The filter function Using our notation a filter function, f, is defined as:

$$f(in_n(CP_i), fs_n(CP_i)) \rightarrow \{out_n(CP_i), fs_{n+1}(CP_i)\} \quad (3)$$

That is, a filter inputs a packet from each child and its current filter state and produces an output packet while updating its local state. f, can be abstracted as two operations: a state join operation, $\sqcup$, which merges states together, and a difference operation, $-$, which computes the incremental difference between two states.

State join The filter uses a function with many properties of a join to update its current state by merging it with the new inputs:

$$in_n(CP_i) \sqcup fs_n(CP_i) \rightarrow fs_{n+1}(CP_i) \quad (4)$$

Based on our experiences, we observe that a join operation with the following properties is not unduly restrictive and admits many useful filters:

Commutative :	$a \sqcup b = b \sqcup a$
Associative :	$(a \sqcup b) \sqcup c = a \sqcup (b \sqcup c)$
Idempotent :	$a \sqcup a = a$

The relevance of commutativity and associativity has been noted previously in the application of parallel prefix computations [27]. These properties enable our relaxed consistency model and facilitate efficient failure recovery. For example, since the order and grouping of operands are irrelevant, we do not have to preserve the original operand order or grouping post-failure.

We can deduce that at any instant a communication process' filter state is the join of the previous inputs it has filtered: $fs_0(CP_i) = \varnothing$, and

$$fs_n(CP_i) = in_0(CP_i) \sqcup \ldots \sqcup in_{n-1}(CP_i) \quad (5)$$

where CP_i has filtered n waves of input.

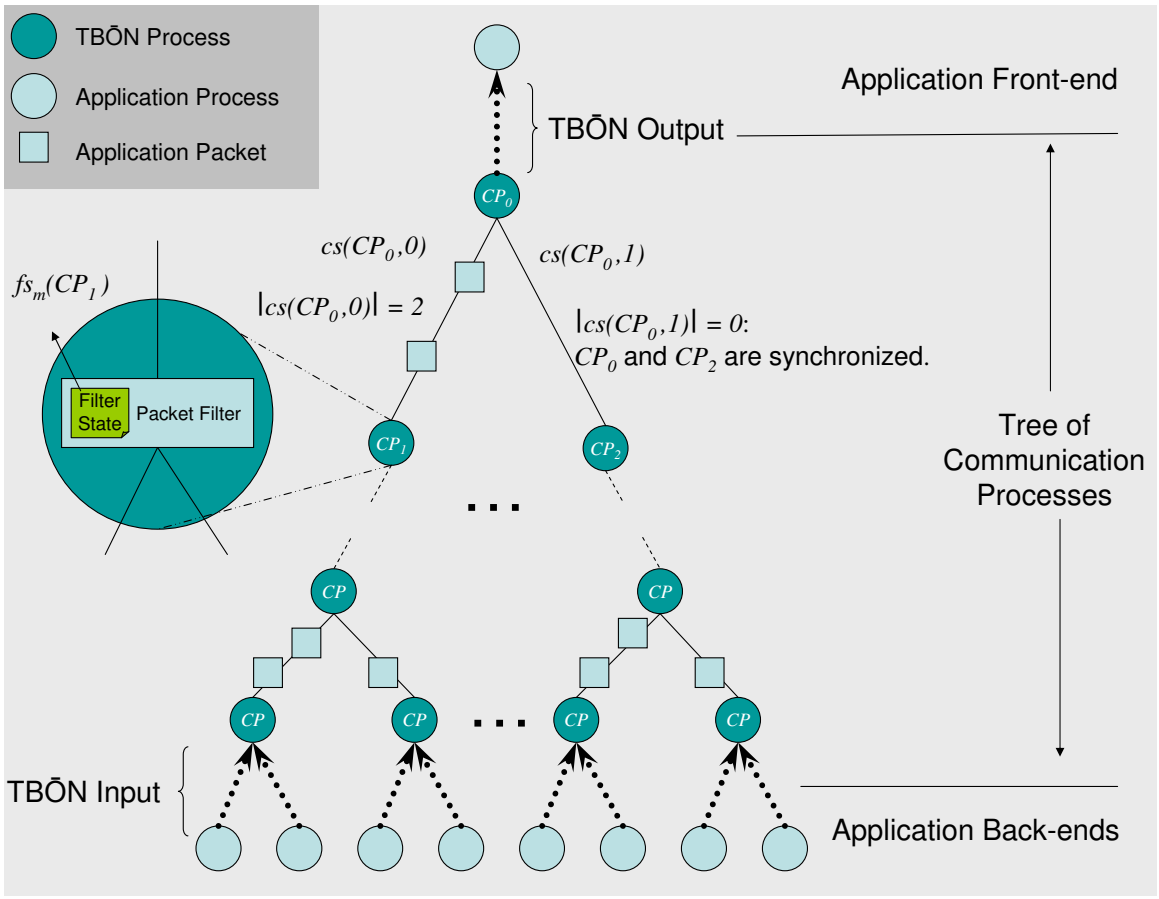

Figure 1. The TBŌN Computational Model. Application back-ends continuously stream data into the TBŌN. Communication processes use filters to aggregate this data and propagate analysis results to the front-end.

State difference The filter function computes its output by calculating the incremental difference between its previous and current states:

$$fs_{n+1}(CP_i) - fs_n(CP_i) = out_n(CP_i) \quad (6)$$

We call "$-$" a *contextual inverse* of $\sqcup$: If $a \sqcup b = c$ and $c - a = b'$, then $a \sqcup b = a \sqcup b'$, and we say $b \equiv b'$ in the context of joining with a. Consider the example where $\sqcup$ is set union, and $-$ is set difference: $\{1,2,3\} \sqcup \{2,3,4\} = \{1,2,3,4\}$, but $\{1,2,3,4\} - \{1,2,3\} = \{4\} \neq \{2,3,4\}$; however $\{1,2,3\} \sqcup \{2,3,4\} = \{1,2,3\} \sqcup \{4\}$.

2.3 The Generality of the TBŌN Model

Previously, we have discussed the tree-based computing model as a powerful abstraction for application performance and scalability [2]. While this previous work did not include a formal specification of the computational model, it contains a fuller discussion of current and potential TBŌN applications. Here, we summarize how these applications map into our formal model, focusing on those stateful filter computations that operate on streams of input (as opposed to a single wave.)

Common stateful data aggregations include simple ones like `historical min`, `historical max`, `count`, and `historical average`, which quickly summarize large data sets. In each of these cases, as new inputs are processed by the filter, they are merged with the current state. For example, `historical max` merges new input by computing the maximum of the new inputs and the previous maximum. The outputs are then computed by calculating the incremental state change, if any. For `historical max`, the previous and new maximums are compared to determine the output update. Such aggregations are used in distributed tools [7, 20], system monitors [36, 38], MANETs [14, 29, 30, 42], and information management systems [5, 32].

More recently, we have been studying more complex data aggregations like clock synchronization algorithms [34], time-aligned data aggregation [34], scalable performance analysis [35], and graph analysis algorithms for visualization [35] and stack trace debugging analysis [1]. Again, these aggregations follow the model of updating current filter state with new inputs and propagating the incremental state changes as output. For instance, in the graph analysis algorithms, new input sub-graphs are folded into the filter's current graph, and the filter outputs the updates to its graph structure.

The TBON model is also well-suited for large classes of analysis algorithms that run on large (terascale and petascale) datasets from scientific and business domains. For instance, data mining or information extraction, the process of distilling specific facts and features from large quantities of data, is the basis for many important application areas including Internet information retrieval [24, 40], bio-information [23], intrusion detection and threat analysis [8], geographical information systems [21], business intelligence [26, 25], and organiz-

ing digital audio collections [39]. These applications can be mapped into the TBON abstraction in two ways: First, typically, some data model or statistical analysis is used to group related elements and distinguish unrelated ones. For dynamic or learning data models, new inputs are used to update the data model and changes to the model are output. Second, these models are used to form data clusters or equivalence classes of input data objects: new inputs are placed into appropriate classes and the incremental changes to the object classifications are output.

3. TBON Theories and Proofs

The following theories and proofs show the relationship between the various filter and channel states throughout the TBON distributed computation. The principle observations are summarized in Theorem 3.3, which states that for any sub-tree, the filter states at that sub-tree's leaf processes contain information that subsumes the sub-tree's channel states, and Theorem 3.4, which states that TBON output is a function of the root process' filter state and the set of TBON channel states. These results inspire our state compensation recovery model, in Section 4, and will be used to prove that this recovery model preserves TBON computational semantics across failures.

This first theorem shows the equivalence between a filter's inputs and its output and is used to support the latter theorems. Intuitively, data aggregation summarizes relevant feature(s) of the input set in its output: output can be considered as the original input minus uninteresting features. For example, a user may only be interested in the maximum temperature readings of a set of sensors for the duration of a simulation; a `max` aggregation would return this result and filter out the other uninteresting temperature readings.

THEOREM 3.1 (Input/output equivalence theorem). *A filter's output is equivalent to its given input in the context of joining with the filter's old state.*

Proof

$$\begin{aligned}
&\text{(Eqn. 4: in} \sqcup \text{fs = fs')}\\
&in_n(CP_p) \sqcup fs_n(CP_p) \quad = \quad fs_{n+1}(CP_p)\\
&\quad - \text{ is inverse of } \sqcup\\
&\quad in_n(CP_p) \quad \equiv \quad fs_{n+1}(CP_p) - fs_n(CP_p)\\
&\text{(Eqn. 6: fs' - fs = output)}\\
&\quad in_n(CP_p) \quad \equiv \quad out_n(CP_p)
\end{aligned}$$

■

Theorem 3.2 formalizes the inherent information redundancy that occurs during a TBON computation as data propagates from the leaves toward the root. This property will be extended to show that the state at the leaf processes of a TBON sub-tree contains all the information available in the rest of that sub-tree.

THEOREM 3.2 (Inherent redundancy theorem). *The join of a communication process' filter state with its pending channel state is equal to the join of its children's states.*

Proof Let CP_i be an arbitrary communication process with child CP_j and CP_k on incoming channels l and m, respectively. Also, let CP_i, CP_j, and CP_k have filtered n, o, and p waves of input, respectively; $n \leq o, n \leq p$.

$$fs_n(CP_i) \sqcup cs_{n,o}(CP_i, l) \sqcup cs_{n,p}(CP_i, m)$$

$$\begin{aligned}
&\text{(Eqn. 5: state = join of input history)}\\
&= in_0(CP_i) \sqcup \ldots \sqcup in_{n-1}(CP_i) \sqcup\\
&\quad cs_{n,o}(CP_i, 1) \sqcup cs_{n,p}(CP_i, m)
\end{aligned}$$

$$\begin{aligned}
&\text{(Eqn. 2: input = join of children's output)}\\
&= (out_0(CP_j) \sqcup out_0(CP_k)) \sqcup \ldots \sqcup\\
&\quad (out_{n-1}(CP_j) \sqcup out_{n-1}(CP_k)) \sqcup\\
&\quad cs_{n,o}(CP_i, j) \sqcup cs_{n,p}(CP_i, k)
\end{aligned}$$

$$\begin{aligned}
&\text{(Eqns. 1 \& 2: channel state = channel source output)}\\
&= (out_0(CP_j) \sqcup out_0(CP_k)) \sqcup \ldots \sqcup\\
&\quad (out_{n-1}(CP_j) \sqcup out_{n-1}(CP_k)) \sqcup\\
&\quad out_n(CP_j) \sqcup \ldots \sqcup (out_o(CP_j) \sqcup\\
&\quad out_n(CP_k) \sqcup \ldots \sqcup (out_p(CP_k)
\end{aligned}$$

$$\begin{aligned}
&\text{(Commuting the operands)} := out_0(CP_j) \sqcup \ldots \sqcup out_o(CP_j) \sqcup\\
&\quad out_0(CP_k) \sqcup \ldots \sqcup out_p(CP_k)
\end{aligned}$$

$$\begin{aligned}
&\text{(Thm. 3.1: output} \equiv \text{input)}\\
&= in_0(CP_j) \sqcup \ldots \sqcup in_o(CP_j) \sqcup\\
&\quad in_0(CP_k) \sqcup \ldots \sqcup in_p(CP_k)
\end{aligned}$$

$$\begin{aligned}
&\text{(Eqn. 5: input history = filter state)}\\
&= fs_o(CP_j) \sqcup fs_p(CP_k)
\end{aligned}$$

■

We can now show that the state at the leaf processes of a sub-tree contains all the information available in the rest of that sub-tree. This means that should any non-leaf channel or filter state be lost, the information necessary to regenerate that state exists at the leaves of any sub-tree that totally contains the lost components.

THEOREM 3.3 (The All-encompassing Leaf States). *The join of the states at the leaves of a TBON sub-tree equals the join of the state at the sub-tree's root process and all the TBON in-flight data.*

Proof We introduce a new operator, $desc^k$, which describes the set of descendants of a communication process k levels away:
$desc^0(CP_i) = CP_i$;
$desc^1(CP_i, j) \rightarrow j^{th}$ child of CP_i;
$desc^1(CP_i) \rightarrow \{desc^1(CP_i, 0), \ldots, desc^1(CP_i, n-1)\}$, $n =$ fanout of CP_i;
$desc(\{CP_m, \ldots, CP_n\}) \rightarrow desc^1(CP_m) \cup \ldots \cup desc^1(CP_n)$; and
$desc^k(CP_i), 1 < k \leq tree\_depth \rightarrow desc(desc^{k-1}(CP_i))$
When the fs and cs operators are used without a subscript, they represent the specified process or channel's current state

based on filtered or incident packets. Similarly, without subscripts, *in* and *out* designate the specified process' input and output history, respectively. Lastly, when any of these operators are applied to a set of processes or channels, they return the join of that operator applied to the individual set's members.

From Theorem 3.2, we deduce:

$$fs(desc^1(CP_0)) = fs(desc^0(CP_0)) \sqcup cs(desc^0(CP_0))$$
$$fs(desc^2(CP_0)) = fs(desc^1(CP_0)) \sqcup cs(desc^1(CP_0))$$
$$\dots$$
$$fs(desc^k(CP_0)) = fs(desc^{k-1}(CP_0)) \sqcup cs(desc^{k-1}(CP_0)).$$

Substituting the former identities into the latter:

$$fs(desc^k(CP_0)) = fs(CP_0) \sqcup cs(desc^0(CP_0)) \sqcup \dots \sqcup cs(desc^{k-1}(CP_0))$$

■

Finally, we can show that the TBON computation's output stream is solely a function of the root process filter state and all the TBON channel states. The TBON input is the stream of inputs filtered by the TBON leaf processes: $in(desc^k(CP_0))$, where k is the TBON depth. The TBON output is the stream of outputs produced at the TBON root process. We define the effective TBON output, $out(CP_0)$, to be the stream of outputs produced by the root process if the system input quiesces and all communication processes become synchronized; that is, the root and the leaf processes have filtered the same number of input waves.

Theorem 3.1 shows the equivalence between a communication process' input and the aggregated output it produces: $in(CP_i) \equiv out(CP_i)$. We can generalize this theorem to state that the join of the inputs of any level of TBON processes are equivalent to the join of the outputs produced by those processes: $in(desc^k(CP_0)) \equiv out(desc^k(CP_0))$. Since output from level k becomes input to level $k-1$, an induction yields:

$$in(desc^k(CP_0)) \equiv out(CP_0) \quad (7)$$

In words, the input to the TBON leaf processes is equivalent to the aggregated output of the root process.

THEOREM 3.4 (TBON Output Theorem). *The output of a TBON computation is solely a function of the TBON root process state and the TBON channel states.*

Proof

$$\begin{aligned} out(CP_0) &\equiv in(desc^k(CP_0)) && \text{(Eqn. 7)} \\ &\equiv fs(desc^k(CP_0)) && \text{(Eqn. 5: input history equals filter state)} \\ &\equiv fs(CP_0) \sqcup cs(desc^0(CP_0)) \sqcup \dots \sqcup cs(desc^{k-1}(CP_0)) && \text{(By Thm. 3.3)} \end{aligned}$$

■

4. State Compensation: A Scalable TBON Failure Recovery Model

In state compensation, we merge states from non-failed TBON processes to compensate for state lost due to process and channel failures. After describing our failure model and recovery guarantees, we define *state composition*, the focal compensation operation in this paper. We use the previous theoretical results to prove that state composition preserves the semantics of the original computation.

4.1 Failure Model

Our state compensation strategy tolerates non-transient, fail-stop failures due to process, node, or network failures assuming we can detect such failures. Performance failures, such as transmission delays, may be treated as network failures. Our approach tolerates such failures at root, leaf, or internal TBON processes.

In large scale environments, there is a legitimate concern for multiple simultaneous failures caused by high failure rates or hardware failures, like that of a network switch or multiprocessor node. We collapse multiple simultaneous failures into *failure zones*, regions of contiguous failed nodes, shown in Figure 2, and use state from processes outside the failure zones for compensation.

For application process failures outside the TBON, application processes may be viewed as sequential data sources and sinks amenable to light-weight, individual checkpointing, which avoids the complexity and cost of distributed protocols. Certain applications may require only a process restart as opposed to a process recovery; for example, a data monitoring application simply may resume sending updated monitor data values. In cases where the application processes use communication channels external to the TBON, distributed checkpoints may be needed – we have not seen real examples where this is necessary.

4.2 Consistency Model

State compensation establishes a recovery guarantee similar to the *convergent* recovery type described by Hwang et al. [22]. Consider the output streams of two identical TBONs executing the same aggregation on the same input stream; one completes without failure, and the other undergoes failure recovery. In convergent recovery, intermediate results may be different, but the final output stream converges to the non-failure case. The temporary output divergence is due to commutations and different associations of input streams that have been re-routed to accommodate the failure(s). Note that convergent recovery preserves all output information, although specific intermediate output data packets may differ.

4.3 State Composition

In Section 3, we demonstrated that: the effective output of a TBON computation is a function of the TBON root's filter state and the TBON channel states; and for any TBON subtree, the filter states at the leaves subsume the rest of the TBON filter and channel states. State composition leverages these two characteristics to recover from process failures by replacing lost channel state with the filter state from the orphaned descendants of processes in a *failure zone*. Specifically, after the orphans are re-adopted into the tree, they propagate their filter state as output to their new parent. We call this *state composition* because the states used for failure compensation form a composite equivalent to the state that has been lost. The fol-

lowing proof shows that it naturally follows that state composition preserves a computation's semantics across failures:

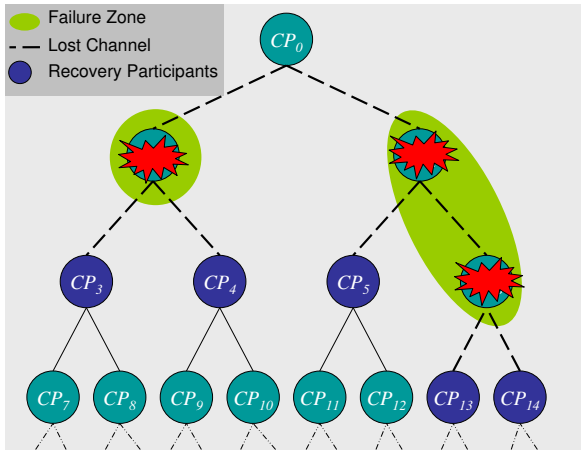

Figure 2. TBŌN *failure zones*, regions of contiguous failures, determine which processes participate in failure recovery.

THEOREM 4.1 (State Composition Theorem). *A TBŌN can tolerate failures without changing the computation's semantics by re-introducing filter state from the descendants of failed processes as channel state.*

Proof Consider a TBŌN, T, with a sub-tree rooted at process CP_i, and let CP_i's m^{th} channel be connected to CP_j, as shown in Figure 3. If CP_j fails, the TBŌN loses the following states: $fs(CP_j)$, $cs(CP_j)$, and $cs(CP_i, m)$. Since Theorem 3.4 states the TBON computation only depends on the system's root and states, we only need to show that re-introducing CP_j's children's state as channel state compensates for the lost states, $cs(CP_j)$, and $cs(CP_i, m)$, i.e. the composition of the former states subsume the latter lost states.

Now, only consider the sub-tree rooted at CP_i and whose only leaves are the children of CP_j. Clearly, this sub-tree's incident channels are the channels lost should CP_j fail. Theorem 3.3 says that filter states at the sub-tree's leaves subsume the states throughout the rest of this sub-tree. Specifically, CP_k and CP_l's states subsume $cs(CP_j)$ and $cs(CP_i, m)$ and can replace those states without changing the computation's semantics. ∎

During normal system operation, we do not explicitly track what messages have been filtered and what output has been transmitted. This means that if the root process fails, we cannot know what output has already been propagated to the application front-end. Therefore, we must act conservatively and regenerate the entire TBŌN output stream. If a new TBŌN process becomes the root, this works automatically as its initial filter state will be empty and as it receives the compensated states from the orphaned processes, its incremental update (from the null state) will be the complete join of the compensating states. If an existing TBON process is promoted to the root, it must propagate its current filter state to the front-end to guarantee that the updates encapsulated in its state are transmitted to the front-end.

While state composition does not tolerate the failure of leaf TBŌN processes, we are studying other compensation operations like state decomposition, which generates compensatory states for a failed child by *extracting* the composition of its

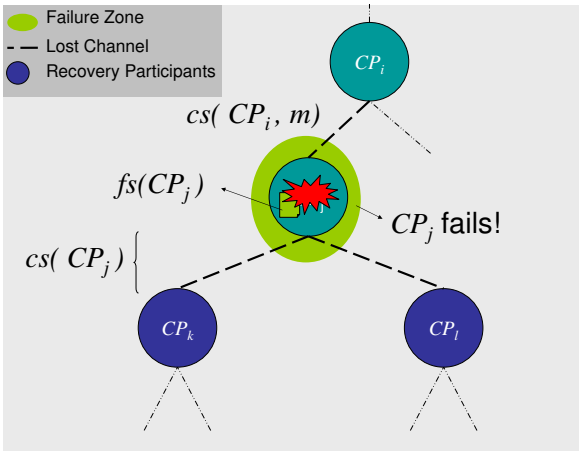

Figure 3. A TBŌN sub-tree with a failed process. When CP_j fails, $fs(CP_j)$, $cs(CP_j)$, and $cs(CP_i, m)$ are lost.

siblings' states from that of its parent. This approach recovers the state of a failed child process to a point synchronized with its parent. However, the recovery of lost in-transit messages may require explicit duplication, like message logging. State decomposition also should prove useful to precisely calculate the (effect of) lost messages and compensate for lost channel state in non-idempotent computations. This will avoid overvaluation by only compensating for lost messages, not those that have been filtered above the failure point. Lastly, our computational model requires that filter functions be commutative and associative. If a heterogeneous set of filters are composed into a single computation, as long as the composition of these filters are commutative and associative, state composition can be leveraged.

5. Failure Recovery Implementation

Our failure recovery has three components: failure detection, process tree reconstruction to accommodate the failure, and lost state compensation. We have developed a fault-tolerant MRNet prototype [34] by adding implementations of these components. An additional outcome of our extensions is that MRNet now supports dynamic changes to its process organization: previously, the MRNet process tree was fixed at startup.

5.1 Event Detection Service

We extended MRNet with an Event Detection Service (EDS) running as a thread within each MRNet process. The EDS' primary purpose is failure detection, however the EDS is also responsible for dynamically establishing new children connections as in the tree reconstruction protocol (discussed in the next section.) In addition, the EDS accepts a special protocol message signaling it to terminate the process. This feature is used to inject failures into the TBON for recovery performance evaluation as described in Section 6.1.

Failure detection in distributed systems comprises component failure detection and failure information dissemination. For component failure detection, we use a lightweight protocol where the EDS at each TBŌN process monitors that process' peers for failures. This approach is scalable as the number of peers that each process monitors is constrained by the process tree's fan-out. In our current implementation, the EDSs at peer processes establish an extra TCP connection used solely for component failure detection. A premature termination of this

connection, for instance due to a host/process crash or a link failure, results in the non-failed kernel aborting the connection, and we deem this a failure of the remote peer. Timeliness of the current detection mechanism depends on the cause of failures: process failures are immediately detected by remote peers. Node failures that prevent the kernel from explicitly aborting connections can be detected via TCP keep-alive probes. Keep-alive probes have a two hour minimum default period that generally can only be lowered on a system-wide basis by privileged users [43]. Heartbeat protocols [] could be used for more responsive node failure detection.

Since each process is monitored by multiple peer EDSs, multiple EDSs will detect the failure of each component. Any EDS that detects a component failure propagates that information to its surviving peers: an EDS that detects that its parent has failed sends that information to its children, and an EDS that detects that one of its children have failed sends that information to its parent and surviving children. Upon receiving failure data, an EDS propagates that data its peers except the one from which it received that data. In this fashion, we leverage the structure of the TBON for scalable failure information dissemination. In the future, we will do a comparative study of TBON-based protocols for information dissemination versus other scalable protocols like epidemic data dissemination [33].

5.2 Tree Reconstruction

Upon failure detection, disconnected subtrees must be reconnected into the main process tree. For tree reconstruction, we consider three things. First, increases in tree depth leads to increases in communication latencies, so we favor connecting disconnected subtrees such that the overall depth of the tree is not increased. Second, overloading parent processes with new subtrees may lead to load imbalances resulting in poor performance, so we desire a mechanism to evenly distribute orphaned processes among adopting parents in the main tree. Third, the TBON may have been organized to place processes according to the physical network, so orphans should favor adoption by parents closer in the virtual topology.

We implement a simple tree reconstruction protocol that adheres to the above issues and can be executed in parallel by each orphaned process. The protocol assumes each orphan has complete topology information but can tolerate outdated information. For each potential adopting process, CP_i, in the main tree, an orphan computes an *adoption weight*, w_i that factors the new tree depth if that process were to adopt that orphan and the proximity of the two processes in the original tree. Adoption weights are positive real numbers, and processes with higher adoption weights are favored for adoption. In a balanced tree when a process fails, orphaned children likely will compute similar adoption weights for the parents in the main tree, since the disconnected subtrees would have similar depths and similar proximities to the potential adopters. This would lead to all orphans favoring the same set of potential adopters and undesired tree imbalances. To mitigate this phenomenon without orphan coordination, we use the adoption weights to perform a weighted random sampling as described by Efraimidis and Spirakis [16]. Briefly, we sort the vector of potential adopters using a key, $k_i = r_i^{\frac{1}{w_i}}$, where r_i is a random number uniformly distributed in $[0,1]$. For any two keys k_i and k_j, $P[k_i > k_j] = \frac{w_i}{w_i+w_j}$. Each orphan iteratively tries to connect to the nodes in the sorted vector until it succeeds in establishing a new parent-child relationship. This allows the protocol to tolerate failures that occur during the recovery process.

After reconstruction, recovery information is disseminated throughout the TBON in a fashion similar to that used for failure reports. A failure report contains the rank of the failed process. A recovery report contains the ranks of the previously orphaned process and its new parent. Since disconnected subtrees remain intact, this information is sufficient for recipients to properly update their topology information. For correctness, we must address both timeliness and consistency issues of our dissemination protocol. As discussed above, orphaned processes tolerate stale topology information by iteratively connecting to potential adopters until this task succeeds. Furthermore, the randomness in our tree reconstruction algorithm helps to mitigate cases where stale data might otherwise lead to overloading a parent process with new children.

For consistency, we must address missing, duplicate, and out-of-order data. Missing data is untimely data with an infinite latency, and the above discussion of timeliness holds. Duplicate reports can be received by a process if its subtree is moved multiple times to different branches of the tree due to multiple failures, and it receives reports from both branches. Applications of failure and recovery reports are idempotent: they amount to a reiteration that a process is still dead or that a child has is still adopted by its parent. Failure reports signal process terminations and therefore cannot contain conflicting information. Out-of-order failure reports then become a timeliness problem and only lead to stale information. On the other hand, multiple recovery reports from multiple failures regarding the same orphan can conflict. If processed in the wrong order, topology information will become incorrect. We adopt the concept of *incarnation versions* to address this problem. Each TBON process maintains an incarnation number. After each recovery, an orphan's incarnation version is incremented and propagated with the recovery report. Processes disregard recovery reports regarding orphans for whom they have received a report with a higher incarnation number.

5.3 Compensating for Lost State

The complete recovery process is implemented as shown in Figure 4: a parent process that detects a child failure removes the failed channel from its list of pending input channels and continues normal operation. Children processes that detect the failure of their parent can no longer propagate filter output toward the TBON root. In the current implementation, orphaned processes temporarily halt new input processing until they are re-integrated into the main tree. Alternatively, orphaned children could continue to fetch and filter new input and buffer its output until it is reconnected to the rest of the tree. Furthermore, since the child process' filter state, which will be propagated toward the root upon child reconnection, would subsume all its channel output, we could continue input filtering and discard all output until failure recovery is complete.

Once the new TBON organization is known, orphaned processes establish a connection with their new parent. After a new parent/child relationship is established, the newly adopted child propagates its current filter state for all active streams as input to the new parent. After this state is re-introduced as channel state, compensation for lost output is complete, and the child process resumes normal input processing.

Failures that occur during the recovery process are easily accommodated. If new failures are detected during the determination of the new tree topology, recovery can be restarted to account for the new failures. If a child is unsuccessful in trying to reconnect to its new parent, it updates its data structures and tries the next process in its sorted potential adopters vector.

```
 1: if detect child failure
 2:         remove failed child from input list
 3:         continue filtering from non-failed children
 4: endif

 5: if detect parent failure
 6:         do
 7:                 determine and connect to new parent
 8:         while failure to connect
 9:         propagate filter state to parent
10: endif
```

Figure 4. Failure Recovery Algorithm

6. Evaluation

Without failures, our state compensation recovery model incurs no overhead. Our evaluation empirically validates the correct operation of state compensation when failures occur and qualifies recovery performance. We perform our tests on a cluster of 3.0 GHz Pentium IV nodes with 1-2GB RAM on a 100Mbit Ethernet. (Comment: Reviewers: We are setting up on Thunder, LLNL's 1,024 node Itanium cluster, to run the larger experiments. We expect them to follow the trends of the smaller experiments.)

6.1 Failure Injection

We have implemented a failure injection service (FIS) that injects failures into the MRNet process tree. The FIS connects to a victim process' EDS and passes it a `terminate_self` event, recording the time the failure was injected. After failure recovery, previously orphaned processes notify the FIS that they have completed recovery. The time the FIS receives recovery information from the last orphan can be used conservatively to compute recovery latencies. This computation is conservative because the latency comprises `terminate self` and `recovery complete` message transmission latencies. Furthermore, recovery information is received sequentially at the FIS allowing for additional serialization overheads.

6.2 Validating the Recovery Model

To verify that state compensation preserves computational semantics across failures, we use an integer union computation. In this computation, application back-ends randomly generate integer elements and propagate them toward the application front-end. Parent processes propagate the unique integers in their input stream filtering out duplicates. Filters at each process keep the set of previously processed integers as persistent state to filter subsequent input. The final output at the application front-end should be the complete set of integers input by the back-ends. The back-ends record their generated data elements to files for comparison with the set output to the front-end. We use the integer union computation as its output is easily verifiable, and this computation is representative of more complex aggregations. For example, the sub-graph folding algorithm [35] used in the Paradyn tool is in essence this computation operating on graph data instead of integral data and using graph merge and graph difference operations instead of integer set union and integer set difference.

After verifying correctness in the absence of failures, we used the FIS to inject failures into running instances of this computation and inspected the result. As expected, the final set generated at the front-end equaled the union of the sets generated by the back-ends. Had data been lost due to the injected failures, the output set at the front-end would be a proper subset of the union of the input sets.

6.3 Micro-benchmarking Failure Recovery

Recall that only orphaned processes (and adopting parents) participate in failure recovery. In this experiment, we study how increased numbers of orphans impact failure recovery latencies. During failure recovery, orphaned processes record the following data: the time to compute its new parent, the time to connect to its new parent, the time to propagate filter state for compensation, and the overall recovery latency. These local measures along with the conservative global recovery estimates computed by the FIS, `GOverall`, are shown in Figure 5 for increasing TBON fan-outs. Our previous MRNet experiences suggest that typical fanouts range from 16 to 32. We (will) test extreme fanouts up to 128 since hardware constraints can force such situations. For instance, LLNL's BlueGene/L prototype enforces a 1:128 fanout from its 1,024 I/O nodes to its 65,536 dual-core compute nodes. A TBON of a depth 3 and a fanout of 64 would be twice the size of this supercomputer. Resource constraints do not allow us to test balanced trees with these large fan-outs, so we organize the test trees such that processes to be killed have the large fan-outs.

In Figure 5, we show the average across all orphans for the locally computed metrics. Since our recovery algorithm does not require coordination and executes in parallel at each orphan, increasing the number of orphans does not increase the time it takes for disconnected sub-trees to be re-integrated into the main TBON tree. In fact, even if we consider the conservative estimate of overall recovery latency computed by the FIS, these latencies of just over 60 milli-seconds for our largest fanout practically constitute no service interuption.

7. Related Work

Reliability protocols can be categorized as fail-over, rollback-recovery, and multi-path data transmission. We include the latter as a state recovery scheme since in aggregation networks, computational state and output data can be synonymous, and thus output-replicating protocols can replicate process state.

In fail-over based schemes, hosts or processes periodically synchronize their states with backup replicas, which replace failed primary components [9, 11, 12]. Component synchronization and high resource utilization during normal operation limit the scalability of such schemes. For example, at a minimum (one backup per primary), fail-over protocols reduce available computational resources by 50%.

In rollback recovery protocols, processes periodically checkpoint their state to persistent storage. Upon failure, the system rolls back the intermediate state in the checkpoint [17]. The common variant of distributed rollback-recovery is coordinated checkpointing, which requires process coordination to record a consistent checkpoint [13, 18]. A recent evaluation of checkpointing for systems projected to be available by 2010 concludes that such protocols will require dedicated resources and that poorly chosen checkpoint intervals may lead to overloaded network and storage resources [19].

In multi-path data transmission protocols, data is disseminated along multiple paths to increase availability should failures occur. This strategy is used in *gossiping protocols* [4, 15, 33] and for multi-path routing protocols in MANETs [14, 30]. Gossip protocols provide probabilistic transmission guarantees with good scalability but are not suitable for applications with high throughput and timeliness requirements. For example, Astrolabe [32], a gossip-based data management system,

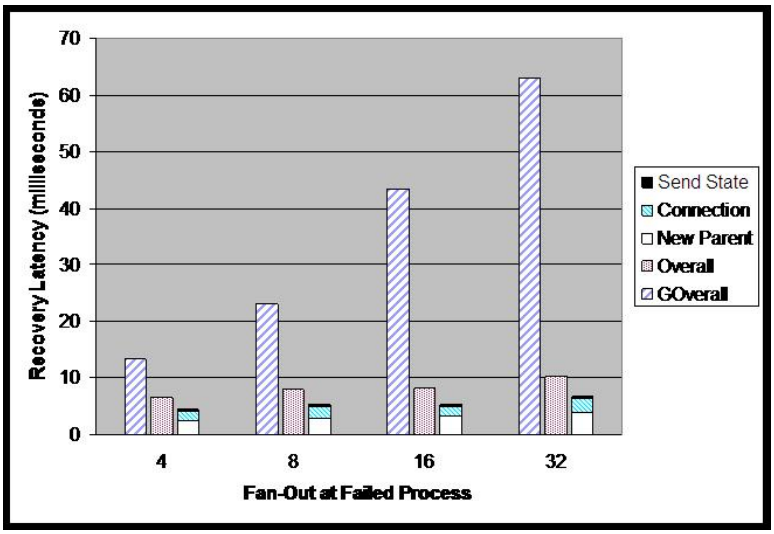

Figure 5. Failure Recovery Micro-benchmark Results. `GOverall`, is a conservative overall recovery estimate recorded by the failure injection service. `Overall` is the average of the total recovery latency recorded by each orphan. `New Parent`, `Connection`, and `Send State` are averages of latencies recorded by each orphan to choose a new parent, connect to the new parent, propagate filter state for compensation, respectively. Note: `Overall` includes other activities like updating local data structures not accounted for in the fine-grained breakdowns.

is designed to provide propagation latencies on the order of tens of seconds and messages are expected to be hundreds of bytes or less. In multi-path routing, the redundant processing of data on different paths can mitigate scalability. The TBON's logarithmic scaling is due to the controlled branching degree; multi-path routing degenerates to a less scalable, though more reliable, lattice structure. Furthermore, with redundant processing *over-counting* must be addressed [30].

In contrast, state compensation provides a scalable solution for TBON computations with deterministic, high throughput data delivery without constraining message sizes or perturbing application performance. Furthermore, failure recovery is fast and only a small subset of processes are interrupted.

8. Conclusion

In response to demands for large scale HPC system reliability, we have developed state compensation, a scalable recovery model for data aggregation in extremely large TBON environments. State compensation exploits the inherent information redundancies in computational states from non-failed processes to generate compensatory states for failed processes avoiding explicit state duplication mechanisms. Furthermore, our recovery protocol does not require process coordination and runs completely independently at orphaned processes. Using our formal model, we have mapped state compensation to a large solution space of important algorithms, and our evaluation shows that we can recover from failures in TBONs with extreme fan-outs representative of terascale and petascale systems in milliseconds. As researchers and developers continue to leverage TBON infrastructures for scalable tools and applications, we believe that no (and low) overhead recovery models like state compensation will prove invaluable. Our plan is to expand this solution space by exploring new compensation operations and studying how current ones apply to extensions of our TBON model. We also plan further investigations of our tree reconstruction process to determine the performance impact of recovered trees and comparisons of TBON information propagation protocols with other protocols for large scale information dissemination.

References

[1] Dorian C. Arnold, Dong H. Ahn, Bronis R. de Supinski, Gregory Lee, Barton P. Miller, and Martin Schulz. Stack trace analysis for large scale applications. In *21st IEEE International Parallel & Distributed Processing Symposium (IPDPS '07)*, Long Beach, CA, March 2007.

[2] Dorian C. Arnold, Gary D. Pack, and Barton P. Miller. Tree-based computing for scalable applications. In *11th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, April 2006.

[3] B.R. Badrinath and Pradeep Sudame. Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. In *9th International Conference on Computer Communications and Networks*, pages 206–213, Las Vegas, NV, October 2000.

[4] Brenda Baker and Robert Shostaky. Gossips and telephones. *Discrete Mathematics*, 2:191–193, 1972.

[5] Magdalena Balazinska, Hari Balakrishnan, Samuel Madden, and Michael Stonebraker. Fault-tolerance in the borealis distributed stream processing system. In *SIGMOD International Conference on Management of Data*, pages 13–24, Baltimore, Maryland, June 2005.

[6] Susanne M. Balle, John Bishop, David LaFrance-Linden, and Howard Rifkin. Ygdrasil: Aggregator network toolkit for the grid, June 2004. Presented in a minisymposium at Workshop on State-of-the-art in Scientific Computing.

[7] Susanne M. Balle, Bevin R. Brett, Chih-Ping Chen, and David LaFrance-Linden. A new approach to parallel debugger architecture. In *Applied Parallel Computing. Advanced Scientific Computing: 6th International Conference, PARA 2002*, Espoo, Finland, June 2002. Published as Lecture Notes in Computer Science 2367, J. Fagerholm et al (Eds), Springer, Heidelberg, Germany, August 2002, pp. 139-149.

[8] Daniel Barbara, editor. *Special Section on Data Mining for Intrusion Detection and Threat Analysis*, volume 30(4) of *ACM SIGMOD Record*, pages 4–64. ACM Press, New York, NY, December 2001.

[9] Joel Bartlett, Jim Gray, and Bob Horst. Fault tolerance in tandem computer systems. In A. Avizienis, editor, *Symposium on the Evolution of Fault-Tolerant Computing*, Baden, Austria, June 1986. Springer Verlag.

[10] Andrew D. Birrell, Roy Levin, Michael D. Schroeder, and Roger M. Needham. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, 1982.

[11] Anita Borg, Jim Baumbach, and Sam Glazer. A message system supporting fault tolerance. In *9th ACM Symposium on Operating System Principles*, pages 90–99, October 1983.

[12] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. Primary-backup protocols: Lower bounds and optimal implementations. In *Third IFIP Conference on Dependable Computing for Critical Applications*, pages 321–343, Mondello, Sicily, September 1992.

[13] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[14] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *20th International Conference on Data Engineering (ICDE '04)*, page 449, Washington, DC, 2004.

[15] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Sixth Annual ACM Symposium on Principles of Distributed Computing(PODC '87)*, pages 1–12, Vancouver, British Columbia, Canada, August 1987.

[16] Pavlos S. Efraimidis and Paul G. Spirakis. Fast parallel weighted random sampling. Technical report, 1999.

[17] Elmootazbellah N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[18] Elmootazbellah N. Elnozahy, David B. Johnson, and Willy Zwaenpoel. The performance of consistent checkpointing. In *11th IEEE Symposium on Reliable Distributed Systems*, Houston, Texas, 1992.

[19] Elmootazbellah N. Elnozahy and James. S. Plank. Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery. *IEEE Transactions on Dependable and Secure Computing*, 1(2):97–108, April-June 2004.

[20] David A. Evensky, Ann C. Gentile, L. Jean Camp, and Robert C. Armstrong. Lilith: Scalable execution of user code for distributed computing. In *6th IEEE International Symposium on High Performance Distributed Computing (HPDC 97)*, pages 306–314, Portland, OR, August 1997.

[21] Ralf Hartmut Guting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–399, 1994.

[22] Jeong-Hyon Hwang, Magdalena Balazinska, Alexander Rasin, Ugur Cetintemel, Michael Stonebraker, and Stan Zdonik. High-availability algorithms for distributed stream processing. In *21st International Conference on Data Engineering (ICDE'05)*, pages 779–790, Tokyo, Japan, April 2005.

[23] Cemil Kirbas and Francis Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys*, 36(2):81–121, 2004.

[24] Mei Kobayashi and Koichi Takeda. Information retrieval on the web. *ACM Computing Surveys*, 32(2):144–173, 2000.

[25] Ron Kohavi and Foster Provost. Applications of data mining to electronic commerce. *Data Mining Knowledge Discovery*, 5(1-2):5–10, 2001.

[26] Ron Kohavi, Neal J. Rothleder, and Evangelos Simoudis. Emerging trends in business analytics. *Communications of the ACM*, 45(8):45–48, 2002.

[27] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, 1980.

[28] Butler W Lampson. Designing a global name service. In *Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC '86)*, pages 1–10, Calgary, Alberta, Canada, August 1986.

[29] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. In *5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[30] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *2nd International Conference on Embedded Networked Sensor Systems(SenSys '04)*, pages 250–262, Baltimore, MD, November 2004.

[31] Katia Obraczka. Multicast transport protocols: A survey and taxonomy. *Communications Magazine, IEEE*, 36(1):94–102, January 1998.

[32] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*, 21(2):164–206, 2003.

[33] Robbert Van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. In *IFIP International Conference Distributed Systems and Platforms and Open Distributed Processing (Middleware 98)*, pages 55–70, The Lake District, England, September 1998.

[34] Phillip C. Roth, Dorian C. Arnold, and Barton P. Miller. MRNet: A software-based multicast/reduction network for scalable tools. In *SC 2003*, Phoenix, AZ, November 2003.

[35] Phillip C. Roth and Barton P. Miller. On-line automated performance diagnosis on thousands of processes. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '06)*, New York, NY, March 2006.

[36] Federico D. Sacerdoti, Mason J. Katz, Matthew L. Massie, and David E. Culler. Wide area cluster monitoring with ganglia. In *IEEE International Conference on Cluster Computing (CLUSTER 2003)*, pages 289–298, Hong Kong, September 2003.

[37] Martin Schulz, Dong Ahn, Andrew Bernat, Bronis R. de Supinski, Steven Y. Ko, Gregory Lee, and Barry Rountree. Scalable dynamic binary instrumentation for blue gene/l. *ACM SIGARCH Computer Architecture News*, 33(5):9–14, 2005.

[38] Matthew J. Sottile and Ronald G. Minnich. Supermon: A high-speed cluster monitoring system. In *IEEE International Conference on Cluster Computing (CLUSTER 2002)*, pages 39–46, Chicago, IL, September 2002.

[39] Wei-Ho Tsai and Hsin-Min Wang. On the extraction of vocal-related information to facilitate the management of popular music collections. In *5th Joint Conference on Digital Libraries (JCDL '05)*, pages 197–206, Denver, CO, 2005.

[40] Luo Xiao, Dieter Wissmann, Michael Brown, and Stephan Jablonski. Information extraction from the web: System and techniques. *Applied Intelligence*, 21(2):195–224, 2004.

[41] Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*, pages 379–390, Portland, Oregon, August/September 2004.

[42] Yong Yao and J. E. Gehrke. Query processing in sensor

networks. In *First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, CA, January 2003.

[43] Victor C. Zandy and Barton P. Miller. Reliable network connections. In *ACM MobiCom*, Atlanta, GA, September 2002.