

Filter Merging for Efficient Information Dissemination

Sasu Tarkoma, Jaakko Kangasharju
{sasutarkoma, jaakko.kangasharju}@hiit.fi



HELSINKI
INSTITUTE FOR
INFORMATION
TECHNOLOGY

Contents

- Introduction
- Preliminaries and Background
- Filter Merging
 - Rules for Merging
 - Integrating Merging with a Routing Table
- Example filter language with merging
- Experimental Results
- Discussion
- Conclusions

Introduction

- Content-based routing has been proposed as a communication primitive for advanced and mobile applications
- Information is propagated from publishers to subscribers (sinks).
- Subscriptions are described using filters. All information sent to a node must match the filters set by the node.
- Content model typically typed tuples or XML.
- Subscription-semantics
 - Primitives: *sub*, *unsub*, *pub*
 - Subscriptions are propagated throughout the system, notifications are sent on the reverse path.
- Advertisement-semantics
 - Optimization to improve scalability.

Background

- Optimizations for scalability
 - Various topologies: hierarchical, peer-to-peer, hybrid, rendezvous-point based (DHT), ..
 - Main idea: propagate the most general set of filters
 - Filter covering
 - Filter merging
- Main research question
 - How to integrate filter merging with a routing table?
- The main contributions of the paper are:
 - Formalizing filter merging in combination with content-based routing
 - Integration with filter covering-based routing tables
 - Initial experimental results with merging

Filters

- Filters typically have two useful properties:
 - Filter F1 is said to cover filter F2 if and only if all the notifications that are matched by F2 are also matched by F1.
 - Overlap is defined similarly and happens when the two filters match the same arbitrary notification.
- The covering relation is a partial order (transitive, anti-symmetric)
- If filters are organized in a graph based on the covering relation, several optimizations are possible.
- The **root set** of the graph is the **minimal cover set**

Data Structures for Cover based Routing

● Filters Poset

- A direct acyclic graph structure that stores the direct predecessors and successors for each filter.
- Two first levels are used to compute the forwards sets.
 - Basic idea: never forward S from X to X if no other neighbour or local client has sent S previously.

● Poset-derived forest

- Data structure with support for fast add/del operations.
- Forest representation based on covering relation
- Local clients, hierarchical, and peer-to-peer routing

● PosetBrowser

- <http://www.hiit.fi/fuego/fc/demos/>

PosetBrowser

The screenshot displays the PosetBrowser application window. On the left is a sidebar with a menu of options: Filters poset (FP), Poset-derived Forest (PF), Balanced Forest (BF), Non-redundant Forest (NRF), a separator, Combined view, Forwards sets (FP,NRF), Filter Merging, another separator, Filter Generation, Execute general tests, and Delete test (FP,NRF). Below the menu is a slider labeled "Number of filters shown" with tick marks at 1, 11, 21, 31, 41, 51, 61, 71, 81, and 91. At the bottom left of the sidebar, it says "Created by Sasu Tarkoma 2005".

The main area of the application is divided into five horizontal panels, each showing a different forest view:

- Filters poset:** Shows a directed acyclic graph with six root nodes at the top and five child nodes at the bottom. The nodes are labeled with expressions like (val1/rn...), (val1/gr...), (val0/rn...), (val0/n...), (val0/n...), (val0/ft ...), (val1/e...), (val0/rn...), (val0/rn...), (val0/e...), and (val0/e...).
- Poset-derived forest:** Shows a similar graph structure to the Filters poset view.
- Balanced forest:** Shows a similar graph structure to the Filters poset view.
- Non-redundant forest:** Shows a similar graph structure to the Filters poset view. The middle child node is highlighted with a green box and contains the text: (val0/rng 17.0 85.0);[iface0],Cl:[iface1, iface0].

Filter Merging

- There are many ways to perform filter merging.
- We assume that a **merge**(F_1, F_2) procedure exists
 - Returns a single merged filter F_M .
 - F_M covers both F_1 and F_2 .
- A merge of two or more filters is called a **merger**.
- Filter merging is useful, because it allows to further remove redundancy and keep the number of elements minimal.
- A merger is either **perfect** or **imperfect**.
 - A perfect merger does not result in false positives or negatives, whereas an imperfect merger may result in false positives.

Filter Merging

- Requirements for filter merging
 - Merging must be transparent for applications and routers.
 - Merging must maintain the set of inserted nodes.
 - An insert of x may result in a new merged node $\text{merge}(x,y)$, but after the delete of x the resulting node must cover y .
- Filter merging may be applied in different places in the event router.
 - local merging,
 - root merging,
 - aggregate merging.
- Two kinds of rules:
 - Mergeability rules, merging rules.

Observations on Routing

- The results are applicable for content-based routing using filter covering.
- Typically, each filter has a **forwards** set and a **subscribers** set.
- All filters that are covered by an active filter from the same interface are removed.
- It follows that the set of active filters for each interface are incomparable.
 - There are no covering relations.
 - A good place to do merging.
- It also follows that the root set of the filters is incomparable.
 - A good place to do merging.

Local merging:
merging for a
particular interface.

Root or aggregate
merging: merging
the root set.

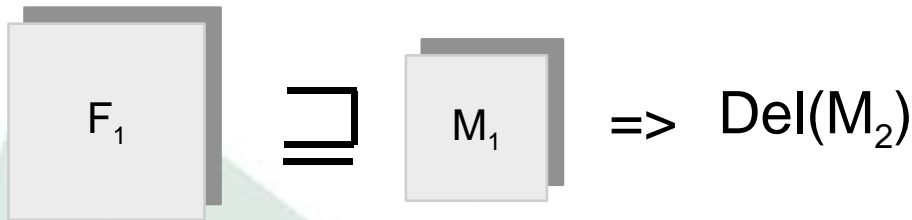
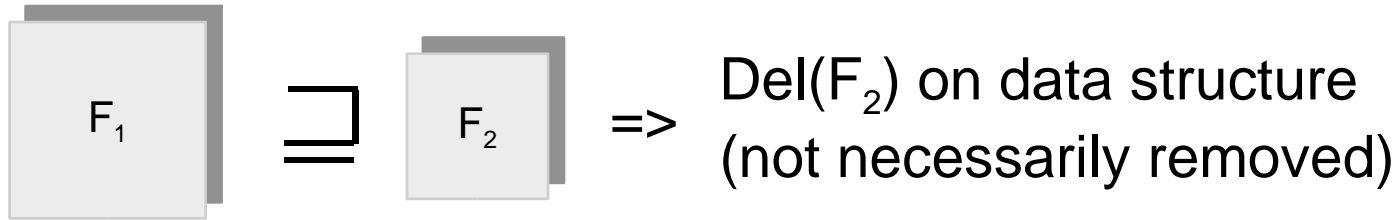
Local Merging

- In **local merging**, filter merging is performed within a data structure.
- Local merging rule:
 - The operation **merge**(F_1, F_2) may be performed if F_1 and F_2 are mergeable and the intersection of their **subscribers** sets has at least one element.
 - The **subscribers** set of the resulting merger must contain only a single element.
- The motivation for local merging is that it benefits the local router. The drawback is that it complicates the data structure.
- In the paper, we focus on **external** filter merging with the root and aggregate merging mechanisms.

External Filter Merging

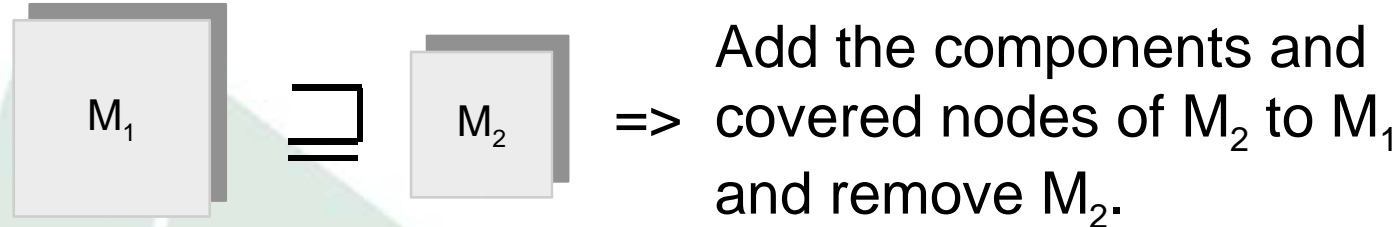
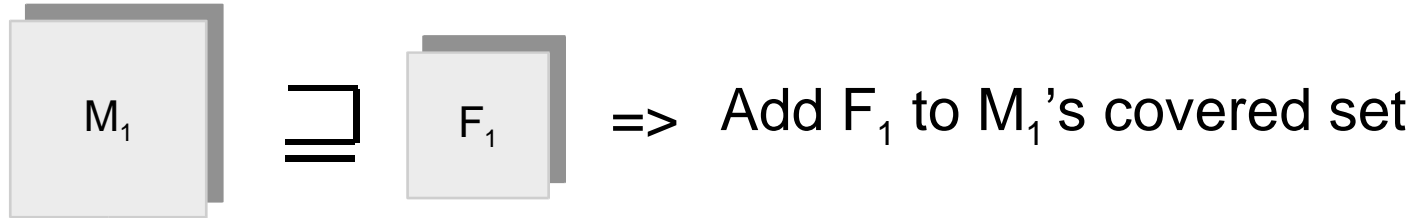
- Basic idea
 - mergers are stored outside the routing table
- Aggregate merging rule:
 - Given that the forwards sets of the filters are non-empty, the filters are mergeable only when forwards $(F_1) \cap \text{forwards}(F_2) \neq \text{empty set}$.
 - The forwards set of the resulting merger is the intersection of the two forwards sets.
- Aggregate filter merging rules are similar to the local merging rules
 - No covered nodes are deleted from the data structure.
 - Direct successors to merged nodes are maintained.
- Add is straightforward. Del is more complicated.
 - Typically adds are more frequent

Rules for Merging I



Del(M_2) \Rightarrow Remove M_2 from MR, reset M_2 's components and covered nodes

Rules for Merging II

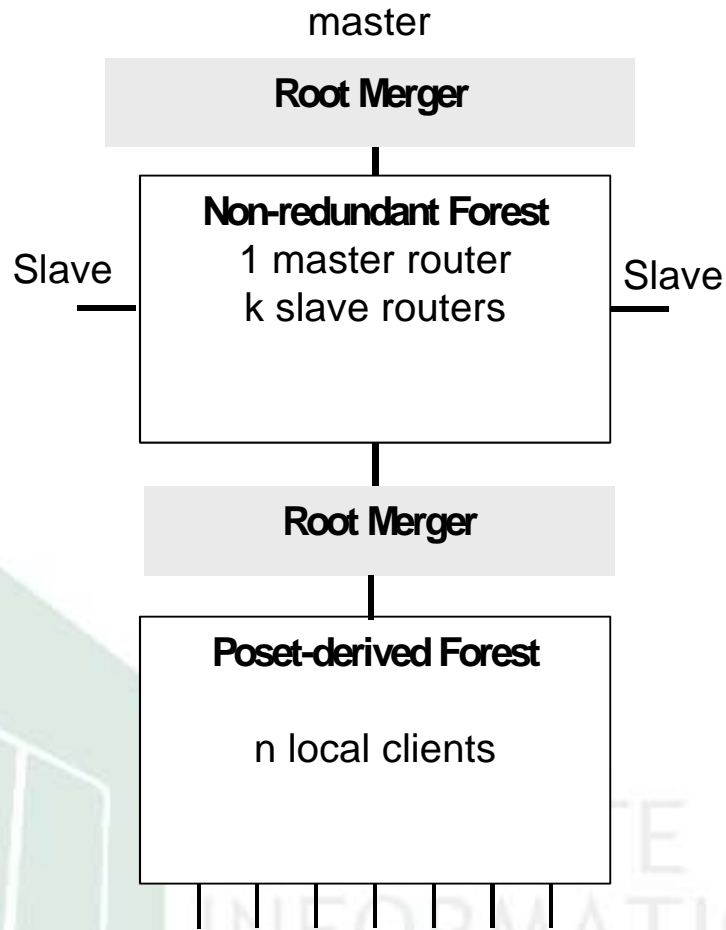


$\text{Del}(x)$ and x is a component of a merger $M_1 \Rightarrow$ Remove M_1 from MR, reset M_1 's components and covered nodes. This makes them available for re-merging.

Integrating Merging with a Routing Table

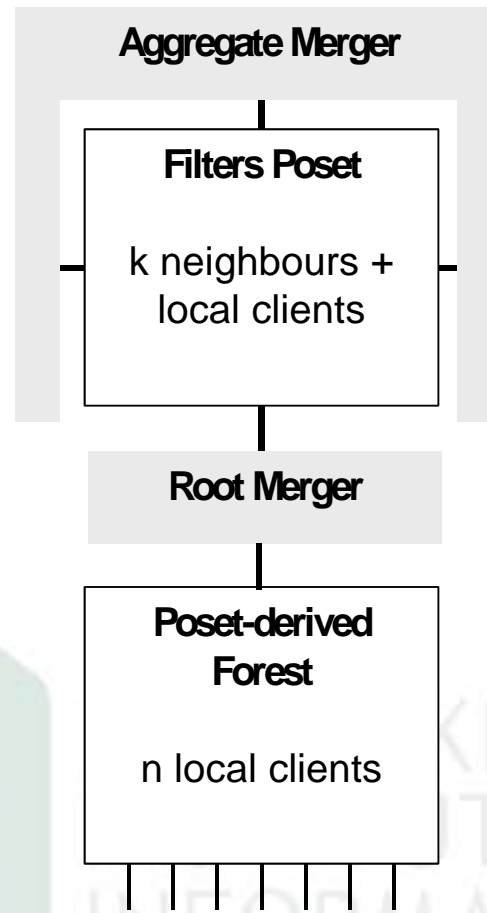
- The proposed forwards set based aggregate mechanism is
 - Generic: it makes minimal assumptions on the underlying data structure.
 - It may be used with both peer-to-peer and hierarchical routing, and also for local clients.
 - Efficient. It is activated only when the root set changes, and it uses the forwards sets to aggregate merger updates. This kind of approach may be used to leverage any multicast mechanisms.
 - Relatively simple. Tracks changes in the root set and merges filters with the same forwards sets. This requires management of the merged sets.

Hierarchical



TECHNOLOGY FOR
INFORMATION
TECHNOLOGY

Peer-to-Peer (non-hierarchical)



Contents

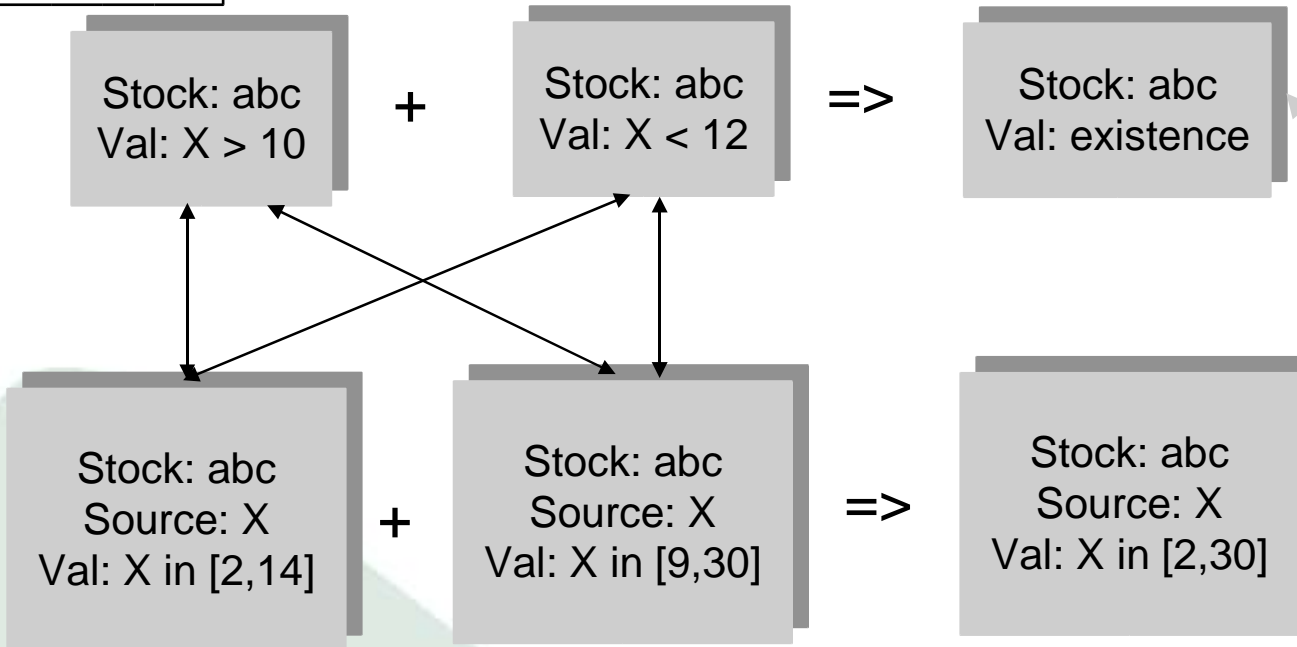
- Introduction
- Preliminaries and Background
- Filter Merging
 - Rules for Merging
 - Integrating Merging with a Routing Table
- **Example filter language with merging**
- Experimental Results
- Discussion
- Conclusions

Filter Language and Merging

- The formal framework supports different filter merging mechanisms.
 - Hardcoded rules, merging procedures, rules from ontologies
 - Merging rules may also be fuzzy
- We use a simple typed tuple model for experiments.
 - Each filter / event conforms to a schema.
 - Each filter is a set of attribute filters with unique name and type.
 - Relational operators are supported.
 - Cover algorithm supports disjunctions (but not conjunctions). Appendix A in the paper.
 - Two filters are mergeable if they have only one differing attribute filter that can be merged.

Filter Merging examples

These four filters do not cover each other

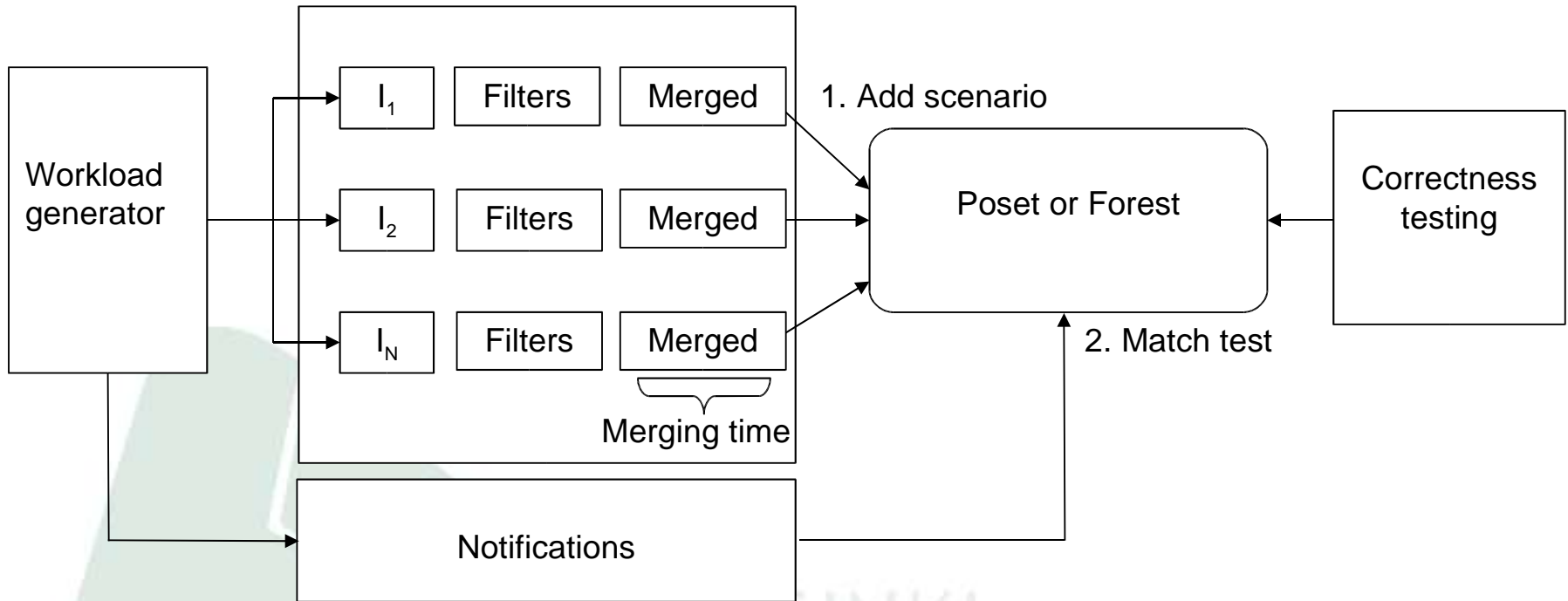


Filter covered by a more simpler one!

Merging is potentially useful given a schema that has good mergeability properties.

Imperfect merging techniques may be used for improved mergeability with the cost of false positives.

Experimental Results I

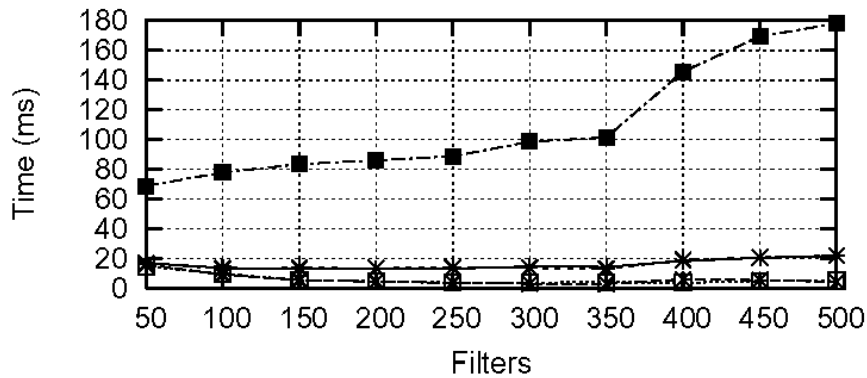


This benchmark corresponds to one-shot set-based merging.

Filter model: variable number of attribute filters (1-3) + type, constraints from the set of relational operators for integers.

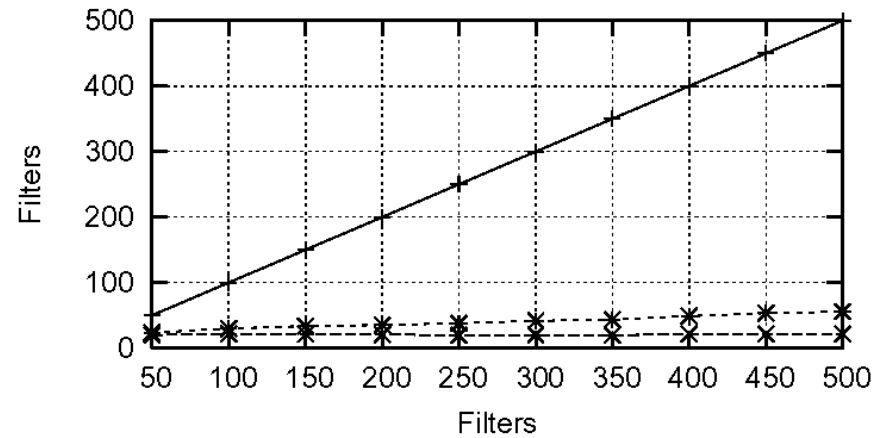
Add with 3 neighbours

Matching time



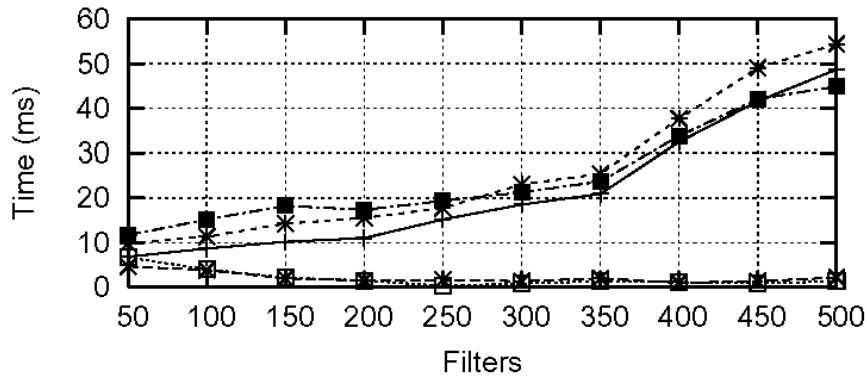
Balanced forest —+—
 Merged balanced forest —x—
 Poset —*—
 Merged Poset —□—
 Naive —■—

Merge set size



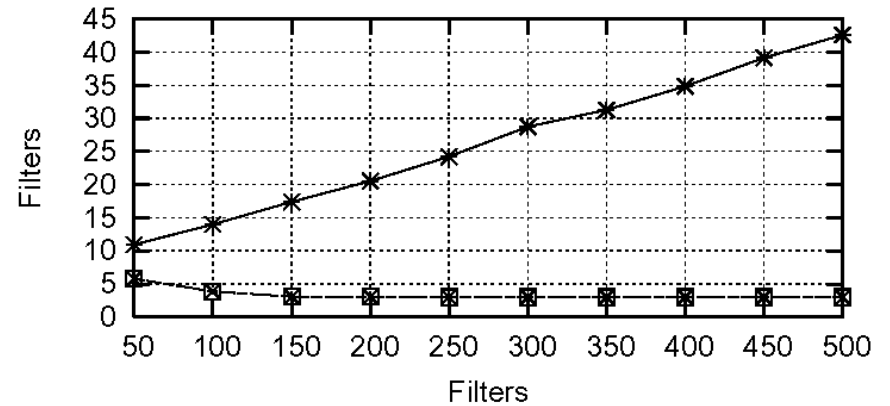
Input set —+—
 Merged set —x—
 Minimal cover set —*—

Add scenario time



Balanced forest —+—
 Merged balanced forest —x—
 Filters poset —*—
 Merged filters poset —□—
 Merging time —■—

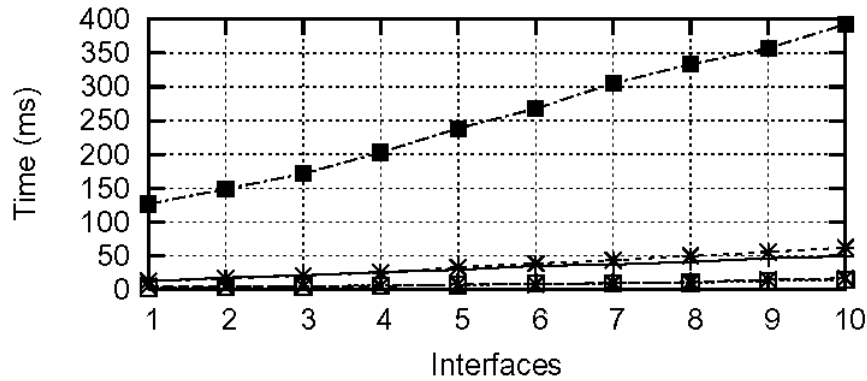
Root size



Balanced Forest root-size —+—
 Merged balanced forest root-size —x—
 Poset root-size —*—
 Merged poset root-size —□—

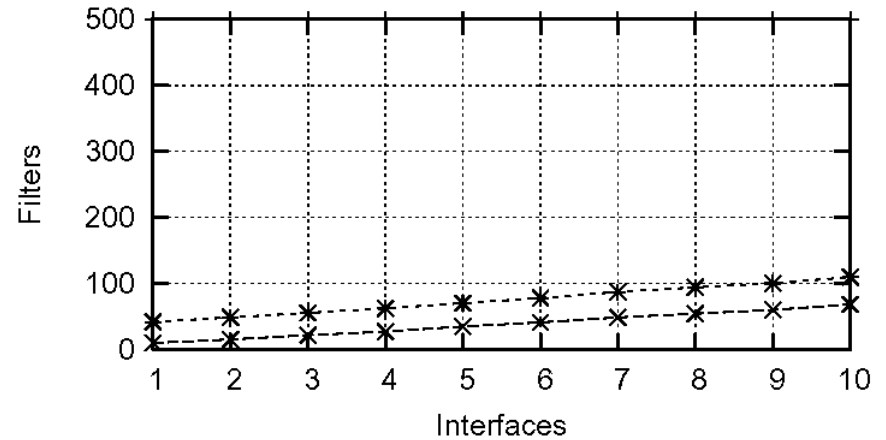
500 filters and a var number of neighbours

Matching time



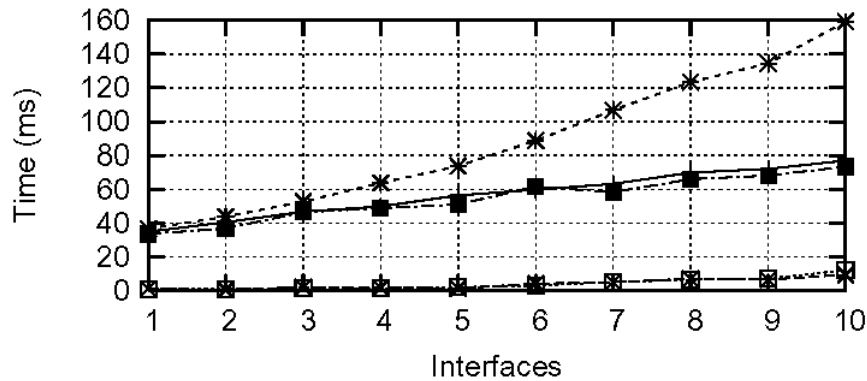
Balanced forest —+—
 Merged balanced forest —x—
 Poset —*—
 Merged Poset —□—
 Naive —■—

Merge set size



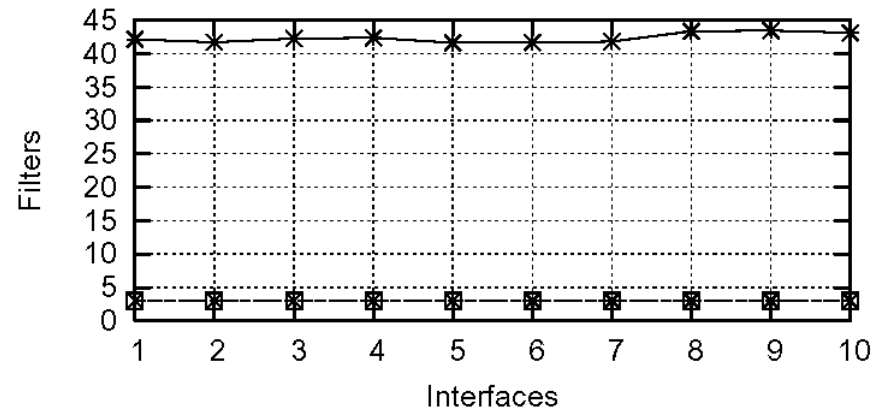
Input set —+—
 Merged set —x—
 Minimal cover set —*—

Add scenario time



Balanced forest —+—
 Merged balanced forest —x—
 Filters poset —*—
 Merged filters poset —□—
 Merging time —■—

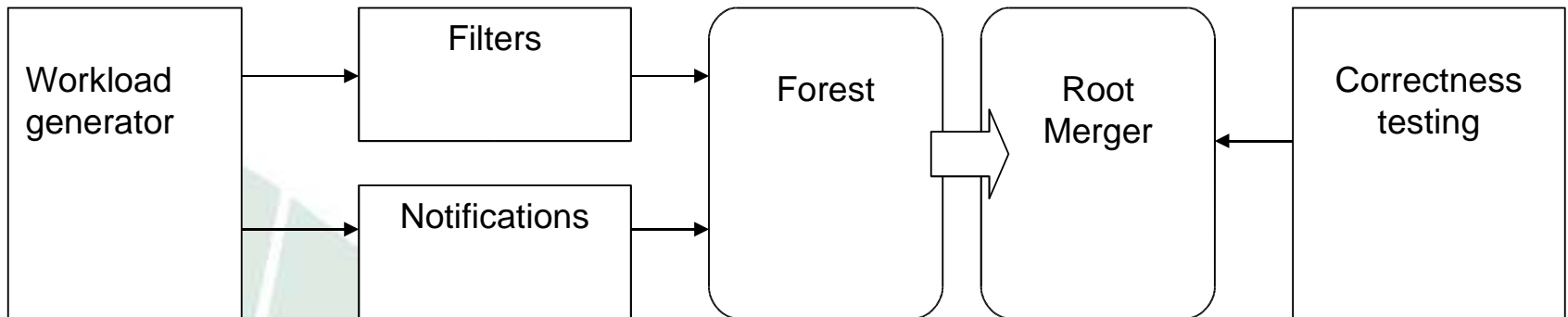
Root size



Balanced forest root-size —+—
 Merged balanced forest root-size —x—
 Poset root-size —*—
 Merged poset root-size —□—

Root Merger Benchmarks

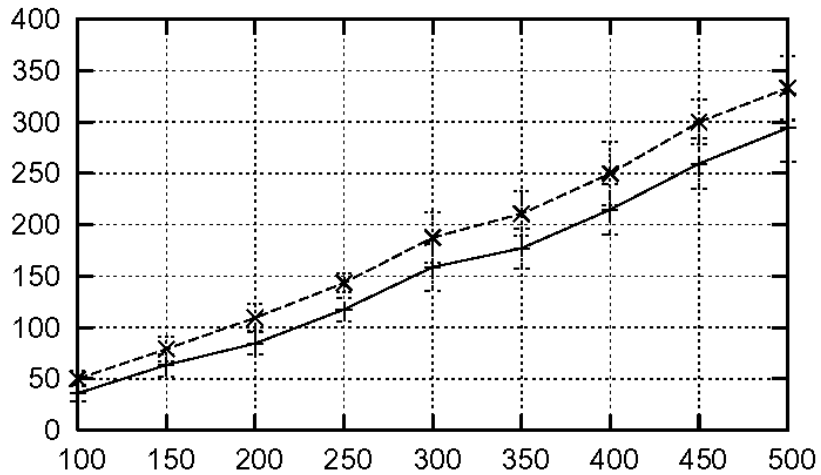
1. Add or Add/Remove scenario



2. Matching test

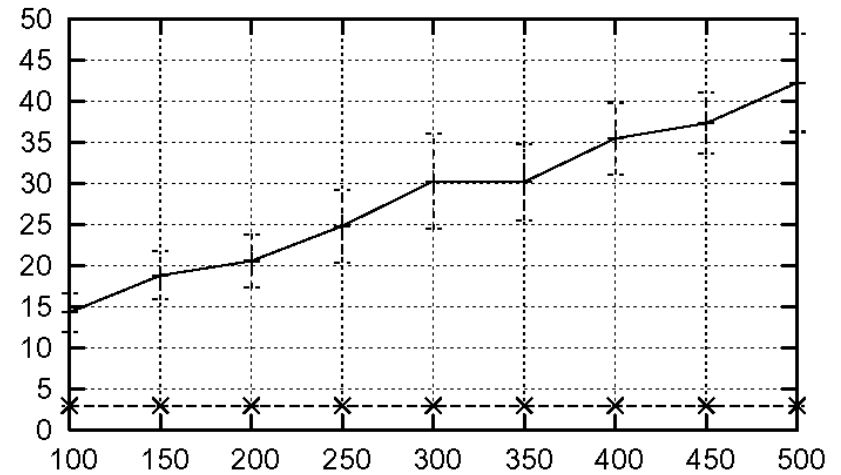
Root merger: Add and Add/Del (3 var AF)

Add scenario time (ms)



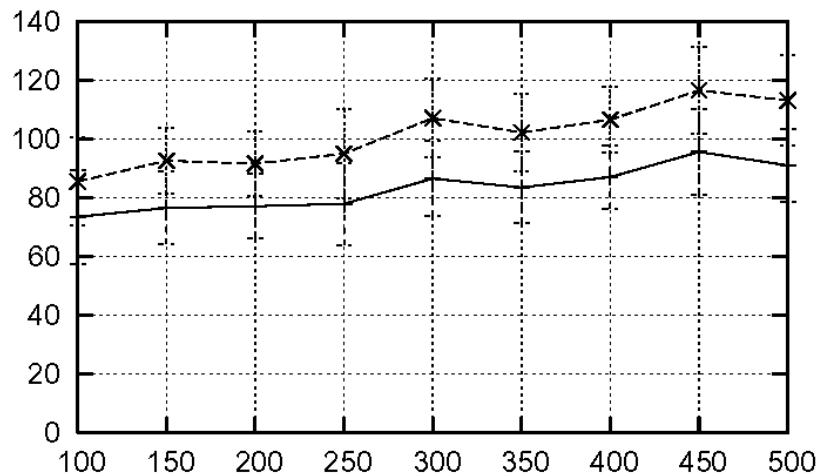
Forest ---+---! Merged forest ---*---!

Add scenario root set size (filters)



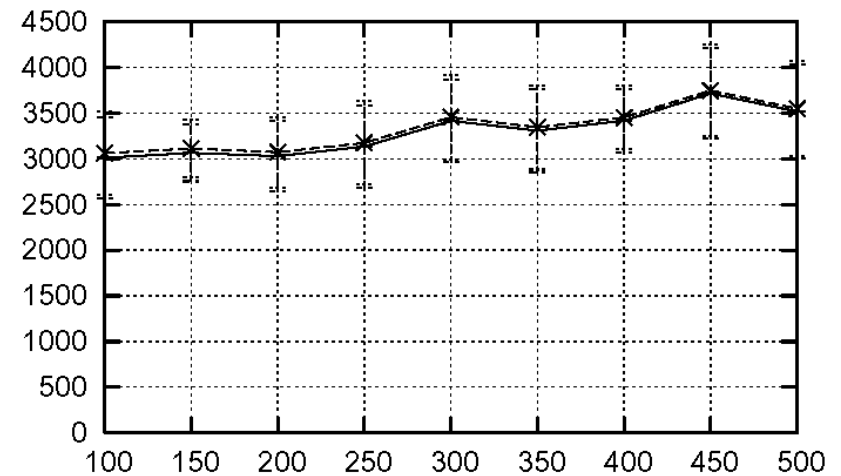
Forest ---+---! Merged forest ---*---!

Add/remove scenario total time (ms)



Forest ---+---! Merged forest ---*---!

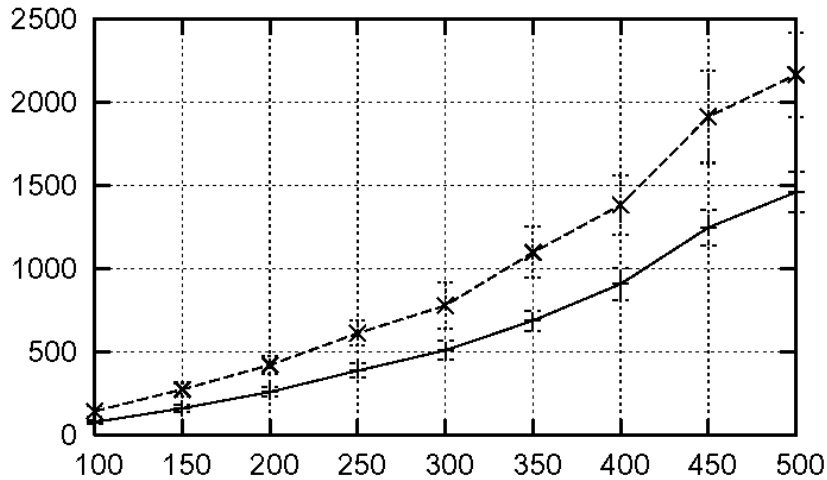
Add/remove scenario total ops



Forest ---+---! Merged forest ---*---!

Root merger: Add and Add/Del (2 AF)

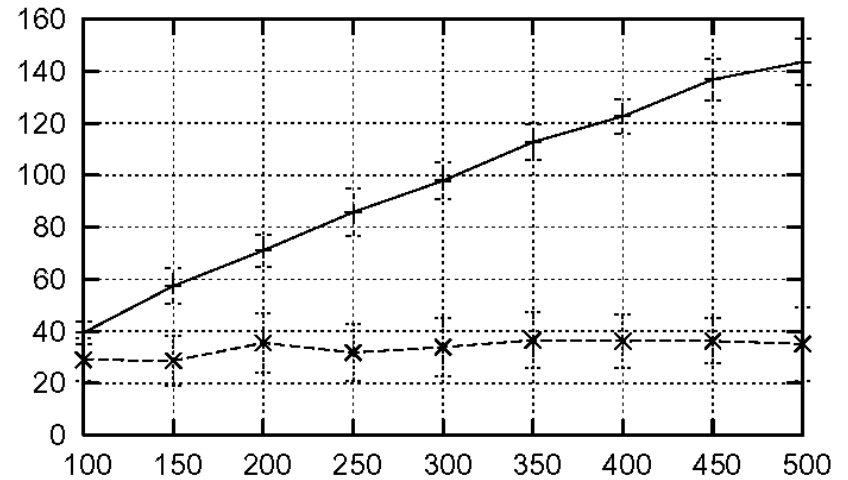
Add scenario time (ms)



Forest !---+---!

Merged forest !---*---!

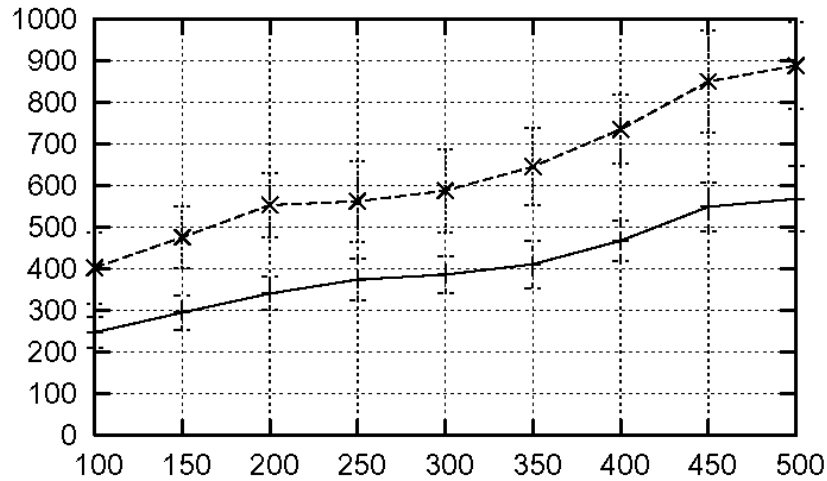
Add scenario root set size (filters)



Forest !---+---!

Merged forest !---*---!

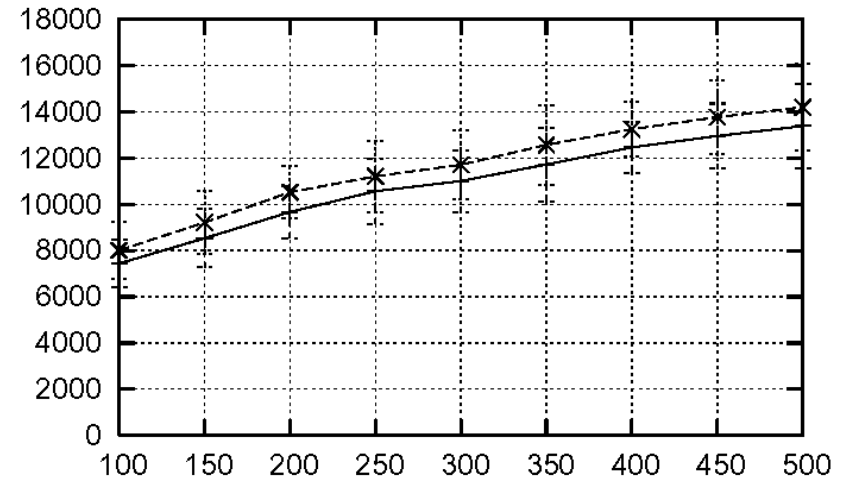
Add/remove scenario total time (ms)



Forest !---+---!

Merged forest !---*---!

Add/remove scenario total ops



Forest !---+---!

Merged forest !---*---!

Discussion

- Results suggest that additions are manageable even in high frequency
- Deletions are problematic
 - Easiest for local clients and hierarchical routing
 - More complicated for peer-to-peer routing
- Strategies in general
 - Results show that some schemas merge well (variable number attribute filters) and some badly (static number of attribute filters)
 - The system should be prevented from trying to merge unmergeable workloads
 - Extend schema to contain mergeability hints
- Strategies for del
 - Lazy evaluation. Monitor false positive rate.
 - Periodic update

Conclusions

- We presented a formal filter merging framework
 - Mergeability and merging rules
 - Independent of the routing structure and filter language
- Integration with content-based routing table based on filter covering
 - Merging the root set
 - Aggregate merging based on the forwards set
 - Cover & merging are useful in combination
- Experimental results indicate that merging is feasible when filters exhibit good mergeability characteristics
 - One-shot merging for periodic updates
 - Runtime merging for dynamic operation
 - Del is challenging, but not expected to be frequent