Efficient Event Correlation over Distributed Systems[1]

Long Cheng, Boudewijn van Dongen, and Wil van der Aalst

Department of Mathematics and Computer Science
Eindhoven University of Technology (TU/e)

[1] Proc. 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2017

Outline

- Background
  - High-performance Analytics.
  - Event correlation.
- Current Approaches
  - Using relational database systems[2].
  - Hashed Values Centric (HVC) using MapReduce[3].
- The Approach: **R**ule **F**iltering and **Gra**ph **P**artitioning (RF-GraP)
- Experimental Results
- Conclusions

[2] Motahari-Nezhad et al. “Event correlation for process discovery from web service interaction logs”. *The VLDB Journal*, 2011

[3] Reguieg et al. “Event correlation analytics: Scaling process mining using MapReduce-aware event correlation discovery techniques”. *IEEE Transactions on Services Computing*, 2015.

Big Data and High-performance Computing

- Big Data:
  - Brings us to a new level of computational complexity: exascale level.
  - Move from a compute-centric to a data-centric model.

Big Data and High-performance Computing

- Big Data:
  - Brings us to a new level of computational complexity: exascale level.
  - Move from a compute-centric to a data-centric model.
- HPC:
  - Programming Languages (e.g., MPI, OpenMP, Chapel, UPC, X10).
  - Computing Frameworks/Platforms (e.g., MapReduce, Spark, HBase, Cassandra, Redis).
  - Computing Infrastructure (e.g., data centers, data warehouse, Cloud).

Performance Challenges

Techniques for parallel/distributed executions rather than the languages or platforms used for implementations are more important.

Performance Challenges

Techniques for parallel/distributed executions rather than the languages or platforms used for implementations are more important.

1. Computation: a very large number of lookup, sort, merge operations would be generated (simplify or reduce such operations).

2. Communication: a very large number of points in the distributed dataset would be potentially accessed (exploit locality of access are required).

3. Load Balancing: real-world data is highly skewed while operations over such data would lead to load imbalancing (remove computation or synchronization hotspots).

Network Communications

Performance of CPUs has grown much faster than network bandwidth in recent years and, as such, the network creates a bottleneck to computation[4].

- data queries could spend more than 65% of their completion time on transferring data over networks.
- data center systems could consume more than 30% of their total energy on communication links, switching and aggregation elements.

[4] Greenberg et al. "VL2: a scalable and flexible data center network", *Communications of the ACM*, 2011

Network Communications

Performance of CPUs has grown much faster than network bandwidth in recent years and, as such, the network creates a bottleneck to computation[4].

- data queries could spend more than 65% of their completion time on transferring data over networks.
- data center systems could consume more than 30% of their total energy on communication links, switching and aggregation elements.

Reducing network traffic is an efficient way to reduce communication time (also others like communication scheduling, routing, etc.).

[4] Greenberg et al. "VL2: a scalable and flexible data center network", *Communications of the ACM*, 2011

Event Correlation

1. Discover a set of correlation rules.
2. Group process events into process instances.

Event Correlation

1. Discover a set of correlation rules.
2. Group process events into process instances.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

- Rule ψ_{A_1,A_1} (i.e., events having the same value on the attribute A_1)
- Several process instances: $\langle e_1, e_2 \rangle$, $\langle e_3 \rangle$ and $\langle e_4 \rangle$.

Event Correlation - Definitions

Definition 1

A *process event log* is a set of events $\mathscr{L} = \{e_1, e_2, ..., e_l\}$, where each event e is represented by a tuple $e_i \in A_1 \times A_2 \times ... \times A_k$, and attributes $A_1, ..., A_k$ represent the union of all the attributes contained in all events. An event typically contains only a subset of these attributes and will therefore have many of its attributes undefined, which will be marked with the value *null*.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Event Correlation - Definitions

Definition 1

A *process event log* is a set of events $\mathscr{L} = \{e_1, e_2, ..., e_l\}$, where each event e is represented by a tuple $e_i \in A_1 \times A_2 \times ... \times A_k$, and attributes $A_1, ..., A_k$ represent the union of all the attributes contained in all events. An event typically contains only a subset of these attributes and will therefore have many of its attributes undefined, which will be marked with the value *null*.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

For example, the event log in above Table has four events and four attributes. For an event $e_x \in \mathscr{L}$, we denote the value of the attribute A_i in the event e_x by $e_x.A_i$.

Definition 2

A defined *correlation condition (or rule)*, is denoted by $\psi(e_x.A_i, e_y.A_j)$, over attributes A_i and A_j of respectively the two events e_x and e_y. If the condition $\psi(e_x.A_i, e_y.A_j)$ returns true then we say that e_x and e_y are correlated through the attributes A_i and A_j.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Definition 2

A defined *correlation condition (or rule)*, is denoted by $\psi(e_x.A_i, e_y.A_j)$, over attributes A_i and A_j of respectively the two events e_x and e_y. If the condition $\psi(e_x.A_i, e_y.A_j)$ returns true then we say that e_x and e_y are correlated through the attributes A_i and A_j.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

There are two kinds of correlation conditions: atomic correlation and multiple correlation conditions (e.g., conjunctive conditions and disjunctive conditions).

Event Correlation - Definitions

Definition 3

An *atomic correlation condition* ψ_{A_i,A_j} specifies that two events are correlated if they have the same value on two of their attributes A_i and A_j, namely, there exists $e_x.A_i = e_y.A_j$ (or $e_x.A_j = e_y.A_i$).

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Definition 3

An *atomic correlation condition* ψ_{A_i,A_j} specifies that two events are correlated if they have the same value on two of their attributes A_i and A_j, namely, there exists $e_x.A_i = e_y.A_j$ (or $e_x.A_j = e_y.A_i$).

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

1. *key-based correlation*: ψ_{A_i,A_j} when $i = j$ (e.g., ψ_{A_1,A_1}).
2. *reference-based correlation*: ψ_{A_i,A_j} when $i \neq j$ (e.g., ψ_{A_1,A_2}).

For presentation, $A_i = A_j$ to replace the condition ψ_{A_i,A_j}

Event Correlation - Definitions

Definition 4

A *process instance* p is a sequence of events corresponding to a subset of events of the log $\mathscr{L}$.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Event Correlation - Definitions

Definition 4

A *process instance* p is a sequence of events corresponding to a subset of events of the log $\mathscr{L}$.

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

For instance, the sequence of $\langle e_1, e_2 \rangle$ is a process instance, if ψ_{A_1,A_1} is a correlation rule.

- Identify the interesting correlation conditions on the basis of observation that some are non-interesting.
- Criterion 1:
  - Key-based conditions: if the value domain of an attribute A_i is very small (e.g., Boolean), then the condition $A_i = A_i$ will be not interesting.
  - Reference-based conditions: if most of the values of an attribute can not be found in another attribute, then the two attributes will be not correlated.

$$distinct\_ratio(A_i) = \frac{|distinct(A_i)|}{|nonNull(A_i)|} \tag{1.1}$$

$$shared\_ratio(\psi_{A_i,A_j}) = \frac{|distinct(A_i) \cap distinct(A_j)|}{Max\{|distinct(A_i)|, |distinct(A_j)|\}} \ (i \neq j) \tag{1.2}$$

  - A threshold α is used to select interesting conditions, $\alpha \leq ratio < 1$.

- Criterion 2: A correlation condition will be not interesting if it partitions a log into large number of short instances or small number of long instances.

$$PI\_ratio(\psi) = \frac{|PI_\psi|}{nonNull(\psi)} \tag{2}$$

  - $|PI_\psi|$ means the number of discovered process instances over the condition ψ, and $nonNull(\psi)$ denotes the number of events for which attributes A_i and A_j of the condition ψ are not null.
  - A threshold β is used to select the interesting conditions, $0.05 \leq PI\_ratio(\psi) \leq \beta$.

Event Correlation - Correlation Discovery

- Compute the properties for each potential correlation rule: *Correlated Message Buffer* (CMB) [3].

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Val		EventSet
C_1	$\rightarrow$	e_3
C_2	$\rightarrow$	e_1, e_2
C_3	$\rightarrow$	e_4

(a) condition $A_1 = A_1$

Val		EventSet1	EventSet2
C_1	$\rightarrow$	e_3	e_1, e_3
C_2	$\rightarrow$	e_1, e_2	e_2, e_4

(b) condition $A_1 = A_2$

Figure: Two CMBs based on applying two correlation conditions over the log.

Event Correlation - Correlation Discovery

- Compute the properties for each potential correlation rule: *Correlated Message Buffer* (CMB) [3].

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

Val		EventSet
C_1	$\rightarrow$	e_3
C_2	$\rightarrow$	e_1, e_2
C_3	$\rightarrow$	e_4

(a) condition $A_1 = A_1$

Val		EventSet1	EventSet2
C_1	$\rightarrow$	e_3	e_1, e_3
C_2	$\rightarrow$	e_1, e_2	e_2, e_4

(b) condition $A_1 = A_2$

Figure: Two CMBs based on applying two correlation conditions over the log.

$$distinct\_ratio(A_1) = \frac{3}{4}, \quad shared\_ratio(\psi_{A_1,A_2}) = \frac{2}{max\{3,2\}} = \frac{2}{3}$$

- For a reference-based case, process instances are computed by applying the DFS (Depth-First Search) algorithm over a bipartite graph, in which the two disjoint vertex sets U and V are represented by the aggregated event sets in a CMB.

EventSet1 (U)	EventSet2 (V)
e_3	e_1, e_3
e_1, e_2	e_2, e_4

Figure: The DFS searching for the CMB with $A_1 = A_2$.

- For a reference-based case, process instances are computed by applying the DFS (Depth-First Search) algorithm over a bipartite graph, in which the two disjoint vertex sets U and V are represented by the aggregated event sets in a CMB.

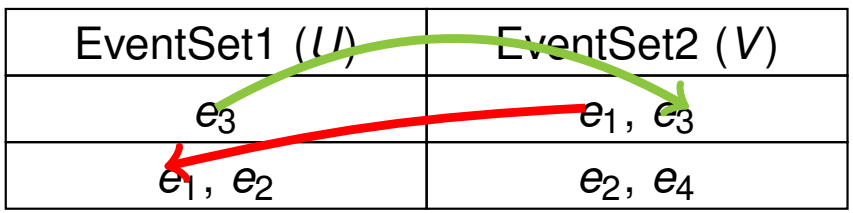


Figure: The DFS searching for the CMB with $A_1 = A_2$.

- For a reference-based case, process instances are computed by applying the DFS (Depth-First Search) algorithm over a bipartite graph, in which the two disjoint vertex sets U and V are represented by the aggregated event sets in a CMB.

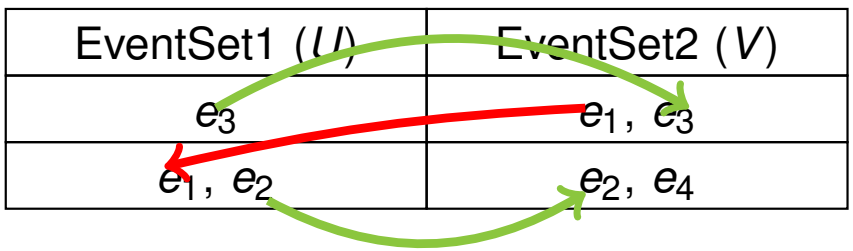


Figure: The DFS searching for the CMB with $A_1 = A_2$.

- For a reference-based case, process instances are computed by applying the DFS (Depth-First Search) algorithm over a bipartite graph, in which the two disjoint vertex sets U and V are represented by the aggregated event sets in a CMB.

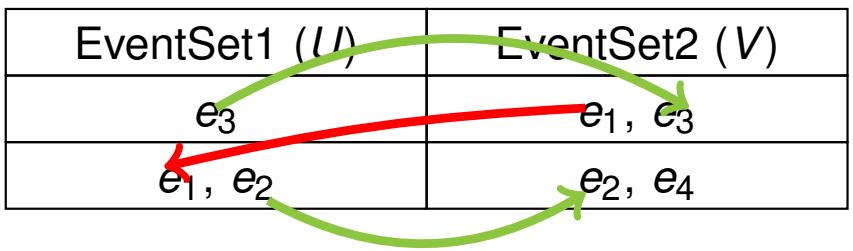


Figure: The DFS searching for the CMB with $A_1 = A_2$.

1. $\{e_3\} \rightarrow \{e_1, e_3\} \rightarrow \{e_1, e_2\} \rightarrow \{e_2, e_4\} \Longrightarrow \langle e_1, e_2, e_3, e_4 \rangle$
2. $PI\_ratio = \frac{1}{4}$ $(shared\_ratio(\psi_{A_1,A_2}) = \frac{2}{3})$
3. If $\alpha = 0.5$ and $\beta = 0.5$, then the condition $A_1 = A_2$ is an interesting rule.

Current Approaches I

- Perform standard SQL queries over a standalone database system to retrieve the CMB related information of a log [2].
- Self-Joins: $A_1 = A_2 \longrightarrow (\mathit{Query}\ 1)$

```
select a.id, b.id
from  ℒ  a,  ℒ  b                    (Query 1)
where a.A1 = b.A2 and b.id > a.id
```

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

- Use all advantages of current database systems on query/join executions (e.g., using indexes).

Current Approaches I

- Perform standard SQL queries over a standalone database system to retrieve the CMB related information of a log [2].
- Self-Joins: $A_1 = A_2 \longrightarrow (Query\ 1)$

```
select a.id, b.id
from ℒ a, ℒ b                    (Query 1)
where a.A1 = b.A2 and b.id > a.id
```

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

- Use all advantages of current database systems on query/join executions (e.g., using indexes).
- Performance issues:
  - number of attributes of a log is large $\rightarrow$ large number of self-joins.
  - large event logs$\rightarrow$ memory and computing cost

Current Approaches II

- *Hashed Values Centric* (HVC) using MapReduce [3]
  1. partition events.
  2. maximize the correlation performance by using parallel implementations.
- Main work flow:
  1. based on all the potential correlation rules, message pairs in the form of *(rule, (value, tag, event))* are generated over all the events on each computing node. The *value* in the pairs means the attribute value and the *tag* is the index number *i* for the attribute A_i.

Current Approaches II

Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

key	(val, tag, event)
$A_1 = A_1$	$(C_2,1,e_1)$, $(C_2,1,e_1)$
$A_1 = A_2$	$(C_2,1,e_1)$, $(C_1,2,e_1)$
$A_1 = A_3$	$(C_2,1,e_1)$, $(C_2,3,e_1)$
$A_1 = A_4$	$(C_2,1,e_1)$, $(C_5,4,e_1)$
$A_2 = A_2$	$(C_1,2,e_1)$,...
$A_2 = A_3$	$(C_1,2,e_1)$,...
$A_2 = A_4$	$(C_1,2,e_1)$,...
$A_3 = A_3$	$(C_2,3,e_1)$,...
$A_3 = A_4$	$(C_2,3,e_1)$,...
$A_4 = A_4$	$(C_5,4,e_1)$,...

(a) message pairs for event e_1

key	(val, tag, event)
$A_1 = A_2$	$(C_2,1,e_2)$, $(C_2,2,e_2)$

(b) $A_1 = A_2$ for event e_2

key	(val, tag, event)
$A_1 = A_2$	$(C_1,1,e_3)$, $(C_1,2,e_3)$

(c) $A_1 = A_2$ for event e_3

key	(val, tag, event)
$A_1 = A_2$	$(C_3,1,e_4)$, $(C_2,2,e_4)$

(d) $A_1 = A_2$ for event e_4

Figure: Generated message pairs by applying candidate correlation conditions on each event in $\mathscr{L}$.

Current Approaches II

- Main work flow:
  2. all the generated message pairs are redistributed to all the nodes based on the hash values of their correlation rules (keys).

Table: Redistribution for message pairs of $A_1 = A_2$

key	(val, tag, event)
$A_1 = A_2$	$(C_2,1,e_1)$, $(C_1,2,e_1)$
$A_1 = A_2$	$(C_2,1,e_2)$, $(C_2,2,e_2)$
$A_1 = A_2$	$(C_1,1,e_3)$, $(C_1,2,e_3)$
$A_1 = A_2$	$(C_3,1,e_4)$, $(C_2,2,e_4)$

  3. local CMBs are built based on the received message pairs and the entire correlation analytics will be terminated when all the CMBs have been processed.

Table: CMB for $A_1 = A_2$

Val		EventSet1	EventSet2
C_1	$\rightarrow$	e_3	e_1, e_3
C_2	$\rightarrow$	e_1, e_2	e_2, e_4

Current Approaches II

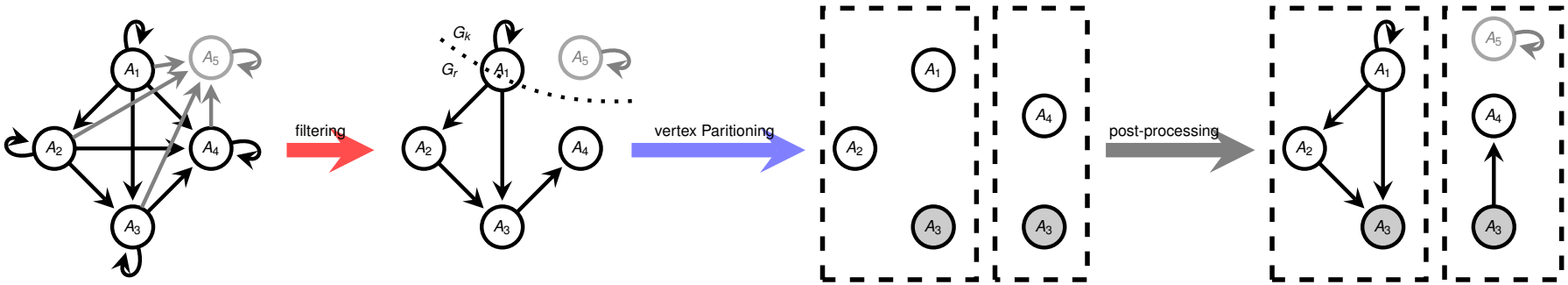
Event	A_1	A_2	A_3	A_4	A_5
e_1	C_2	C_1	C_2	C_5	C_6
e_2	C_2	C_2	C_4	C_4	C_6
e_3	C_1	C_1	C_2	C_5	C_7
e_4	C_3	C_2	C_4	C_4	C_6

- HVC can be parallelized across nodes, the scheme offers the potential for scalability and thus can be applied on large logs.
- Performance issues:
  - within the *generating-and-pruning* scheme.
  - the number of generated message pairs could be very huge $\rightarrow$ network communication and computing is expensive ($10 \times 2 \times 4 = 80$ message pairs for $\mathscr{L}$).

The Proposed Approach (RF-GraP)

- Follow the *filtering-and-verification* scheme:
  1. Filter phase: incorporate a light-weight filter unit into all possible correlation rules to prune large number of non-interesting rules without significantly increasing processing time.
  2. Verification phase: use a graph partitioning approach to decompose the potentially correlated events into chunks by exploring data partitioning and locality assignment.

The Proposed Approach (RF-GraP)

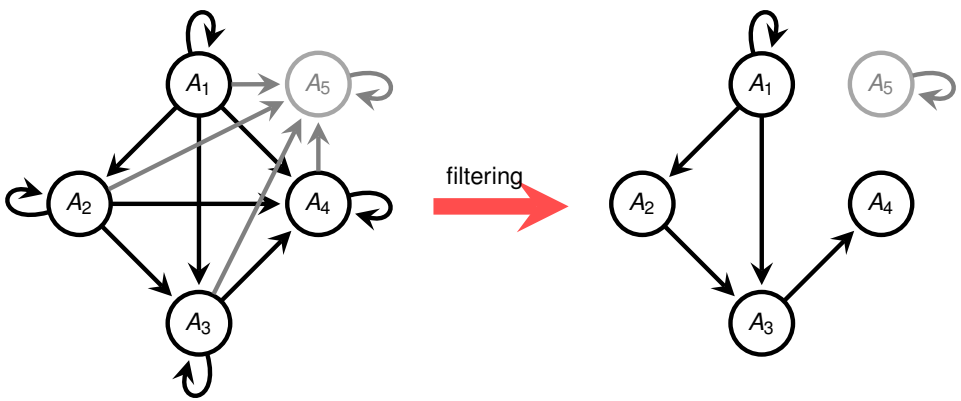


(a) a correlation graph G_c (b) the simplified graph G_s (c) partitioning result of graph (d) solution for data assignment

- Main work flow:
  1. Model all the possible correlation conditions of a log as a correlation graph.
  2. Simplify the correlation graph by filtering out potential interesting rules*.
  3. Partition the simplified correlation graph by exploring data locality*.
  4. Generate message pairs based on the partitioned result and then transfer them to the responsible nodes*.
  5. Build local data structures (e.g., CMBs) based on the allocated message pairs at each node.
  6. Compute the process instances and calculate the values of $PI\_ratio(\psi)$ to get the final outputs.

Rule Filtering in RF-GraP

- Filter out candidate correlations based on the first criterion: $distinct\_ratio(A_i)$ and $shared\_ratio(\psi_{A_i,A_j})$.
- Only one MapReduce job, similar as WordCount() example, but with attributes. For example, for the attribute A_1,
  - Map: $(A_1,(C_2,1)),(A_1,(C_2,1)),(A_1,(C_1,1)),(A_1,(C_3,1))$
  - Reduce: $A_1 \rightarrow (C_2,2),(C_1,1),(C_3,1) \Longrightarrow distinct\_ratio(A_1) = \frac{3}{4}$



(a) a correlation graph G_c (b) the simplified graph G_s

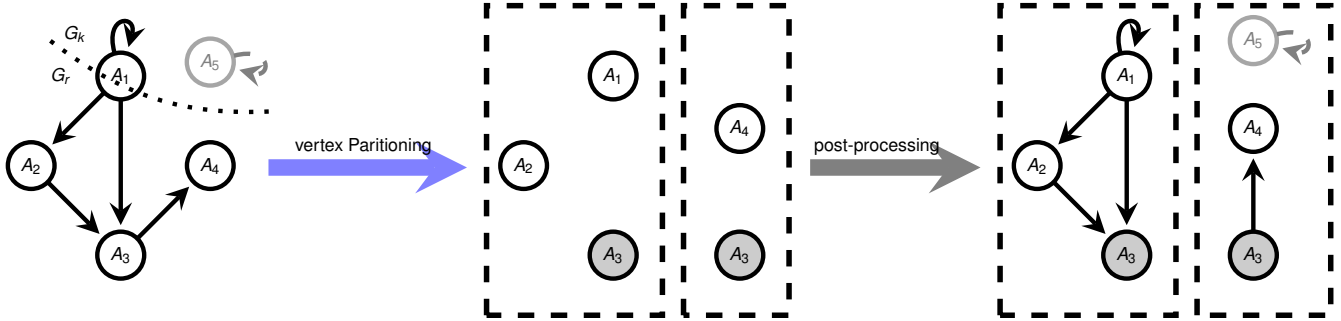
Figure: with $\alpha = 0.7$ for $\mathscr{L}$

Graph Partitioning in RF-GraP

- Hash-based partitioning could bring in redundant message pairs that need to be transferred over networks.

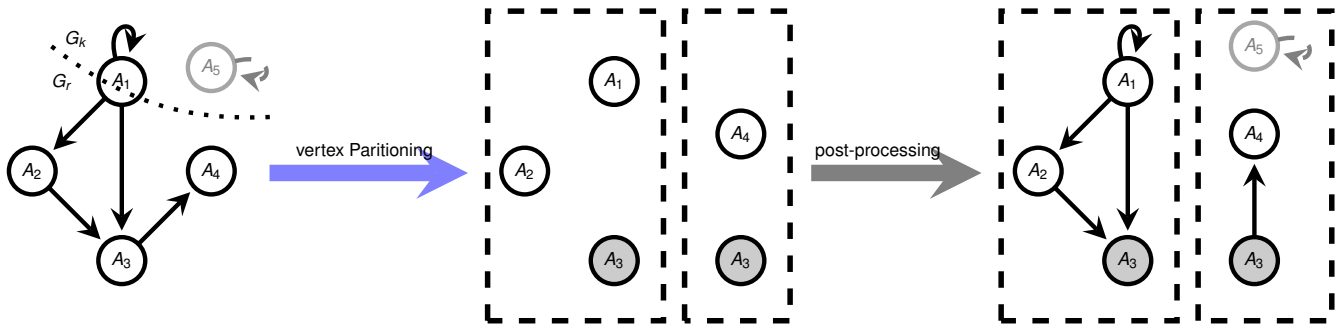
key	(val, tag, event)
$A_1 = A_1$	$(C_2,1,e_1)$, $(C_2,1,e_1)$
$A_1 = A_2$	$(C_2,1,e_1)$, $(C_1,2,e_1)$
$A_1 = A_3$	$(C_2,1,e_1)$, $(C_2,3,e_1)$
$A_1 = A_4$	$(C_2,1,e_1)$, $(C_5,4,e_1)$
$A_2 = A_2$	$(C_1,2,e_1)$,...
$A_2 = A_3$	$(C_1,2,e_1)$,...
$A_2 = A_4$	$(C_1,2,e_1)$,...
$A_3 = A_3$	$(C_2,3,e_1)$,...
$A_3 = A_4$	$(C_2,3,e_1)$,...
$A_4 = A_4$	$(C_5,4,e_1)$,...

Figure: Generated message pairs for e_1



(b) the simplified graph G_s (c) partitioning result of graph (d) solution for data assignment

Graph Partitioning in RF-GraP

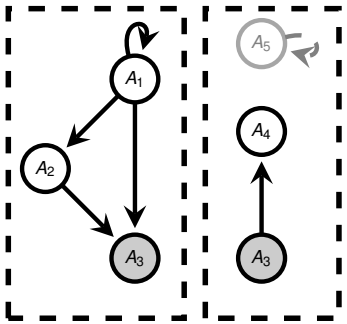


(b) the simplified graph G_s (c) partitioning result of graph (d) solution for data assignment

- The simplified graph G_s is composed of two parts, the subgraph G_r from the reference-based conditions ψ_r, and G_k from the key-based ψ_k.
  1. divide vertexes in G_r into disjoint partitions using vertex partitioning (using METIS partitioner), and assign the vertexes to partitions without destructing their connections in G_r.
  2. for a vertex partitioning, it is very possible that large number of vertexes closed to each other will be located to a same partition, and consequently results in load balancing problems. To improve this, we assign the element in G_k to all the partitioned subgraphs to balance the workloads (we can use more advanced *balanced graph partitioning*[5]).

[5] K. Andreev and H. Racke, "Balanced graph partitioning", *Theory of Computing Systems*, 2006.

Pair Generation and Assignment in RF-GraP



- Generate message pairs based on *attributes* rather than *rules* $\Longrightarrow$ remove redundant pairs.
- *Comprehensive Correlation Message Buffer* (C^2MB) $\Longrightarrow$ reduce memory cost.

key	(val, tag, event)
A_1	(C_2,1,e_1)
A_2	(C_1,2,e_1)
A_3	(C_3,3,e_1)
...	...

(a) generated pairs for e_1

Val		EventSet1	E-Set2	E-Set3
C_1	$\rightarrow$	e_3	e_1, e_3	$\emptyset$
C_2	$\rightarrow$	e_1, e_2	e_2, e_4	e_1, e_3
C_3	$\rightarrow$	e_4	$\emptyset$	$\emptyset$
C_4	$\rightarrow$	$\emptyset$	$\emptyset$	e_2, e_4

(b) the proposed C^2MB data structure

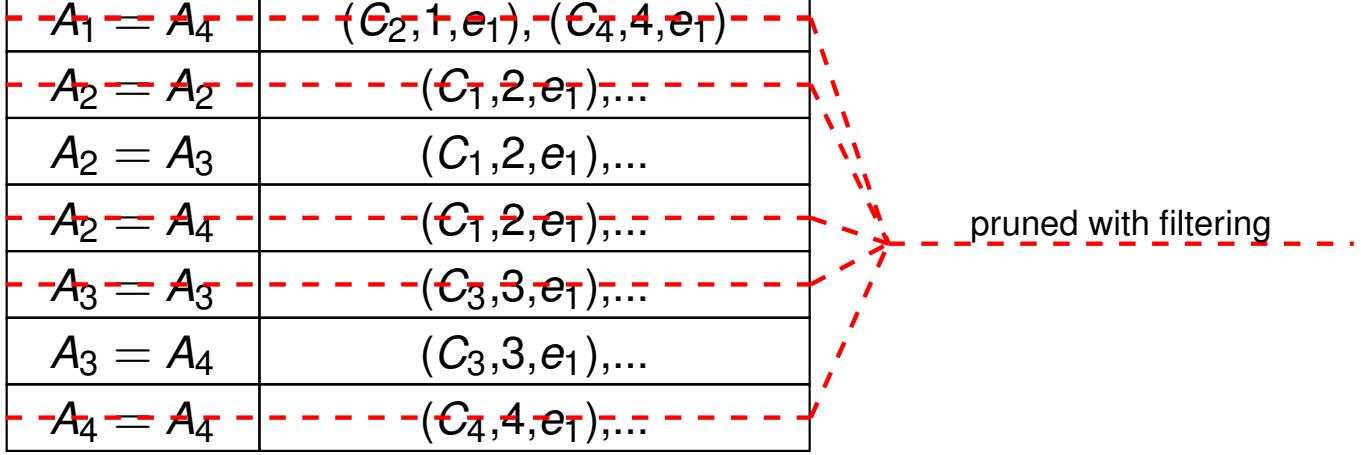
Figure: The form of the generated message pairs, and the data structure used for event correlations in RF-GraP.

Advantages of RF-GraP

key	(val, tag, event)
$A_1 = A_1$	(C_2,1,e_1), (C_2,1,e_1)
$A_1 = A_2$	(C_2,1,e_1), (C_1,2,e_1)
$A_1 = A_3$	(C_2,1,e_1), (C_3,3,e_1)
$A_1 = A_4$	(C_2,1,e_1), (C_4,4,e_1)
$A_2 = A_2$	(C_1,2,e_1),...
$A_2 = A_3$	(C_1,2,e_1),...
$A_2 = A_4$	(C_1,2,e_1),...
$A_3 = A_3$	(C_3,3,e_1),...
$A_3 = A_4$	(C_3,3,e_1),...
$A_4 = A_4$	(C_4,4,e_1),...

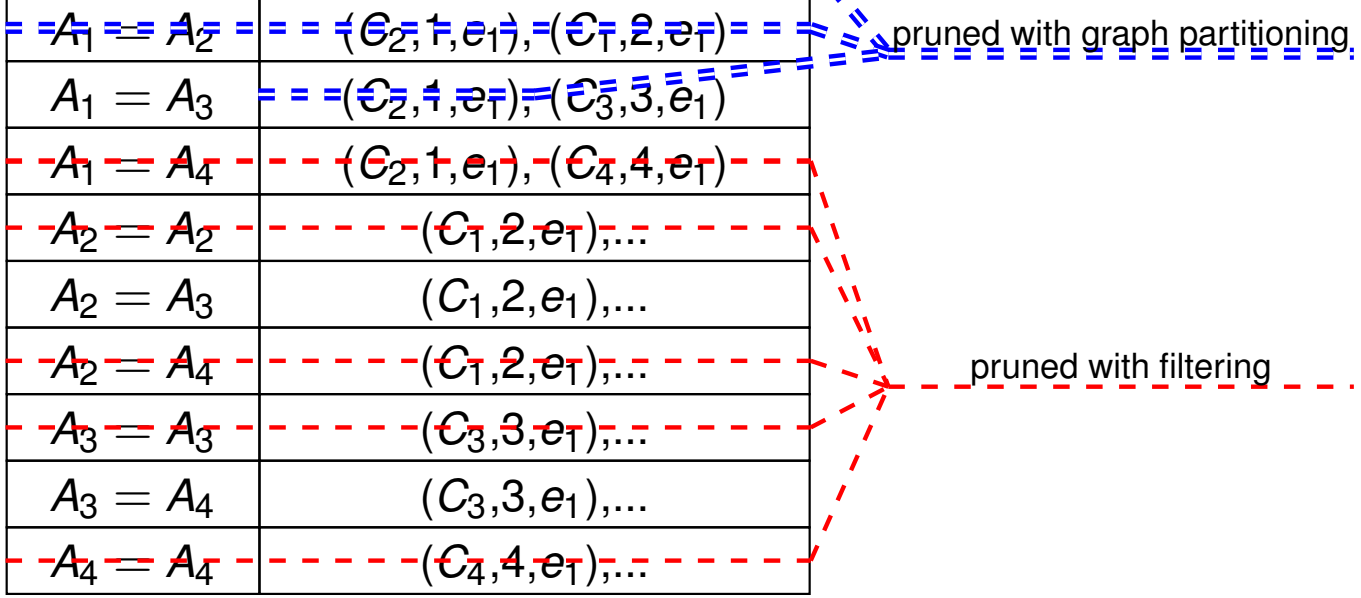
Advantages of RF-GraP

key	(val, tag, event)
$A_1 = A_1$	$(C_2,1,e_1)$, $(C_2,1,e_1)$
$A_1 = A_2$	$(C_2,1,e_1)$, $(C_1,2,e_1)$
$A_1 = A_3$	$(C_2,1,e_1)$, $(C_3,3,e_1)$
~~$A_1 = A_4$~~	~~$(C_2,1,e_1)$, $(C_4,4,e_1)$~~
~~$A_2 = A_2$~~	~~$(C_1,2,e_1)$,...~~
$A_2 = A_3$	$(C_1,2,e_1)$,...
~~$A_2 = A_4$~~	~~$(C_1,2,e_1)$,...~~
~~$A_3 = A_3$~~	~~$(C_3,3,e_1)$,...~~
$A_3 = A_4$	$(C_3,3,e_1)$,...
~~$A_4 = A_4$~~	~~$(C_4,4,e_1)$,...~~

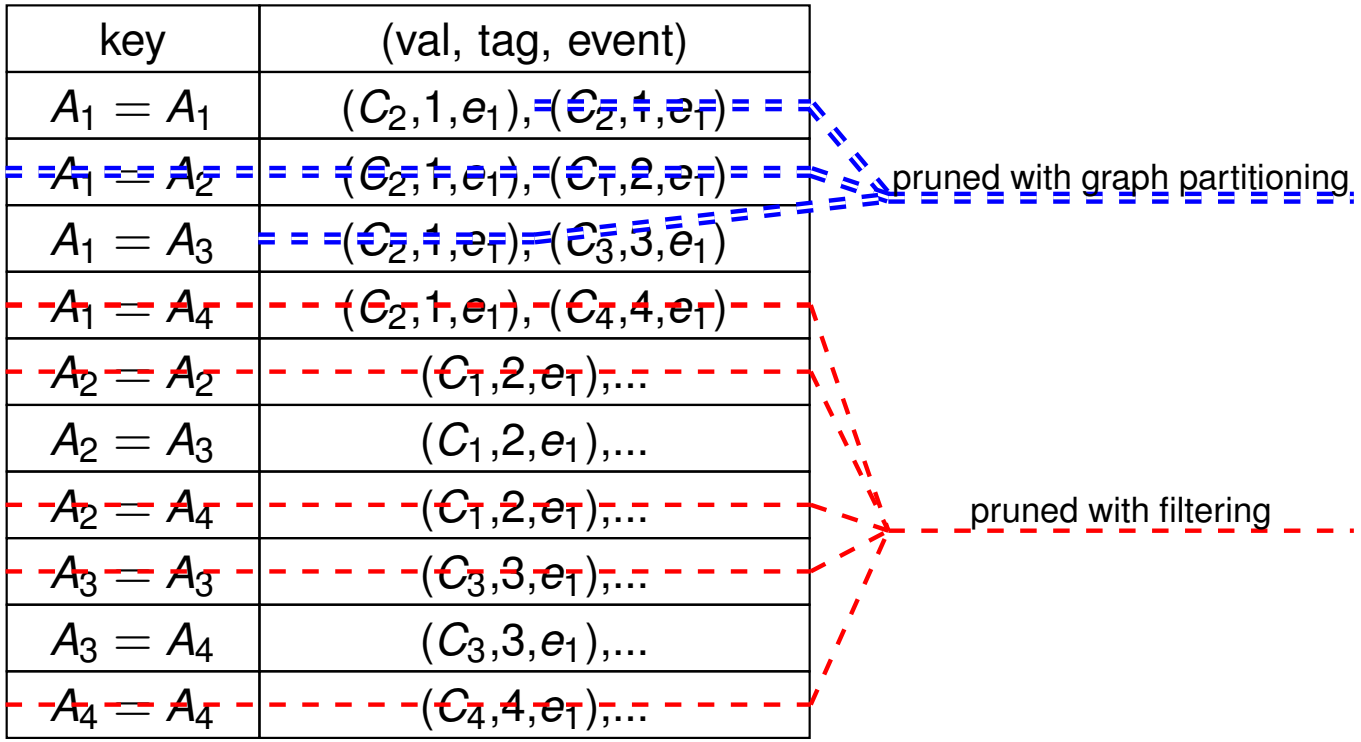


Advantages of RF-GraP

key	(val, tag, event)
$A_1 = A_1$	$(C_2,1,e_1),(C_2,1,e_1)$
$A_1 = A_2$	$(C_2,1,e_1),(C_1,2,e_1)$
$A_1 = A_3$	$(C_2,1,e_1),(C_3,3,e_1)$
$A_1 = A_4$	$(C_2,1,e_1),(C_4,4,e_1)$
$A_2 = A_2$	$(C_1,2,e_1),...$
$A_2 = A_3$	$(C_1,2,e_1),...$
$A_2 = A_4$	$(C_1,2,e_1),...$
$A_3 = A_3$	$(C_3,3,e_1),...$
$A_3 = A_4$	$(C_3,3,e_1),...$
$A_4 = A_4$	$(C_4,4,e_1),...$



Advantages of RF-GraP



- Generated message pairs are highly reduced $\Longrightarrow$ computing and network communication cost $\downarrow$.
- $C^2MB \Longrightarrow$ memory consumption $\downarrow$.
- Designed for distributed environments $\Longrightarrow$ big event data.

Experimental Frame

- Platform:
  1. each node contains 4 CPU cores running at 2.80 GHz with 32GB of RAM.
  2. nodes are connected by Infiniband.
  3. Spark 2.0.0, Hadoop 2.7.3, Scala 2.11.4, Java 1.7.0 25, and Metis 5.1.0.
- Datasets:
  1. quality of discovered rules over a real log: event log from Xerox.
  2. performance tests on an event log extracted from the SCM business service. (a) from [2], [3]; (b) the original log has 19 attributes and 4,050 events; (c) increased the number of events and attributes by a factor of 250 and 2 (around 1 million events and 38 attributes).
- Setup
  1. general Spark and HDFS configurations.
  2. threshold α, varying from 1% to 10%.
  3. threshold $\beta = 0.5$ (based on suggestions).
- Codes are available at https://github.com/longcheng11/ECA.

Quality of Event Correlations

- Based on *correlation matrices*.
- The accuracy of a discovered rule will be the fraction of marked values in its responsible correlation matrix that are correct, compared to the ideal one.
- The higher a value is, the better the rule will be.

0	Event ID	0
1	Start_End	start
2	ITE_ID	40842566
3	TAS_ID	3
4	TAS_NAME	OCRFlowValve
5	STACKNAME	STACK_9_201510300837
6	BATCHNAME	BATCH_4_15103003596
7	BATCHNAME_2	BATCH_4
8	DCN	151030808370
9	ITS_STATUS	2
10	STATUS	A
11	ITS_DTSTART	2015-11-01 21:15:09.040
12	IMAGECOUNT	12
13	PAGECOUNT	12
14	PROCESSDATE	2015-10-30 00:00:00
15	USE_LOGIN	MrAuto
16	KEYLOCATION	Cebu

(a) attribute names and example values

correlated	e_1	e_2	e_3	e_4
e_1	1	1	0	0
e_2	1	1	0	0
e_3	0	0	1	1
e_4	0	0	1	1

(b) matrix for the ideal condition

correlated	e_1	e_2	e_3	e_4
e_1	1	1	1	1
e_2	1	1	1	1
e_3	1	1	1	1
e_4	1	1	1	1

(c) matrix for the rule $A_1 = A_2$

Figure: The attribute names of the Xerox log we used in our quality validation, and example correlation matrices.

Quality of Event Correlations

- We have the knowledge that `DCN` is actually the ideal condition (i.e., $(8, 8)$ is the idea case).

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(a) $\alpha = 0.01$ and $\beta = 0.5$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(b) $\alpha = 0.001$ and $\beta = 0.5$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(c) $\alpha = 0.01$ and $\beta = 0.8$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(d) $\alpha = 0.001$ and $\beta = 0.8$

Figure: Correlation accuracy over the Xerox log.

Quality of Event Correlations

- We have the knowledge that `DCN` is actually the ideal condition (i.e., $(8, 8)$ is the idea case).

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(a) $\alpha = 0.01$ and $\beta = 0.5$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(b) $\alpha = 0.001$ and $\beta = 0.5$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

(c) $\alpha = 0.01$ and $\beta = 0.8$

condition	score
(8,8)	1.0
(6,6)	0.9995673
(2,2)	0.9995673

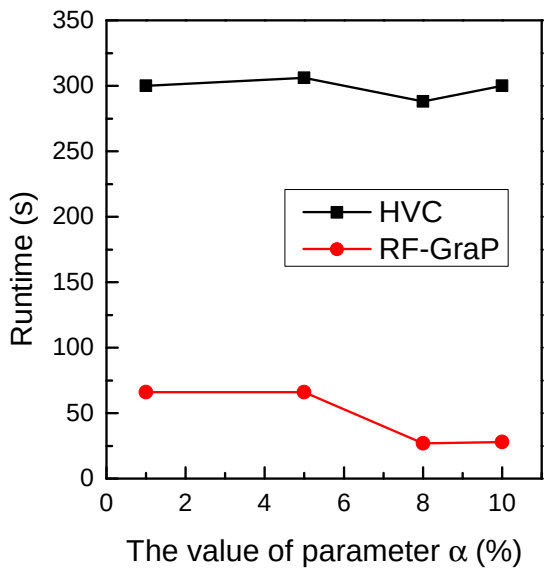
(d) $\alpha = 0.001$ and $\beta = 0.8$

Figure: Correlation accuracy over the Xerox log.

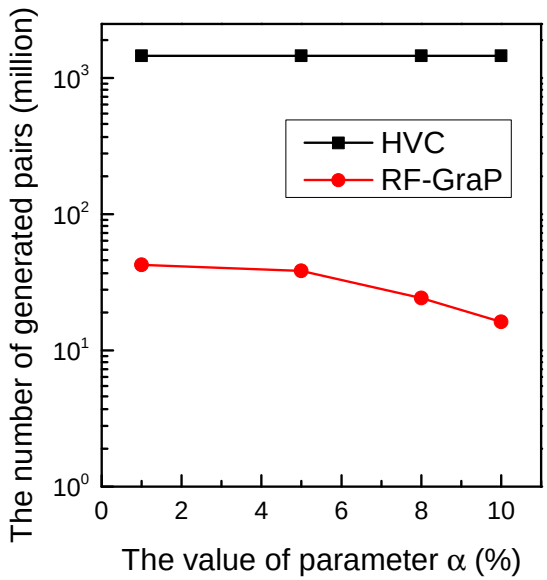
- Accuracy of the rules `(6,6)` and `(2,2)` is very close to 1: since the `ITE_ID` and `BATCHNAME` are the upper level attributes of events.
- Initial results suggest that the correlation approach we adopted is applicable also in real-life settings.

Performance Results - Efficiency

- RF-GraP performs much faster then the HVC in all the cases, and is able to achieve a speedup of $4.5 - 10.7\times$.
- with increasing α, the runtime of HVC is generally the same while RF-GraP decreases.
- The number of *gmp* of RF-GraP is much less than HVC.



(a) runtime efficiency



(b) # message pairs (log-scale)

Figure: The efficiency of each algorithm over the default dataset, with varying the values α (using 32 cores).

Performance Results - Cardinality Experiments

1. Fix the attributes of the log, to 38, and vary the number of events from 0.5 million to 2 million.
2. Fix the number of events to 1 millions, and vary the number of attributes from 19 to 76.

α	1%			10%		
Scale	0.5	1	2	0.5	1	2
Speedup	5.1	4.5	>6.6	8.0	10.7	>44.7

(a) varying the scale factor of number of events

α	1%			10%		
Scale	0.5	1	2	0.5	1	2
Speedup	1.6	4.5	>50.7	5.2	10.7	>61.3

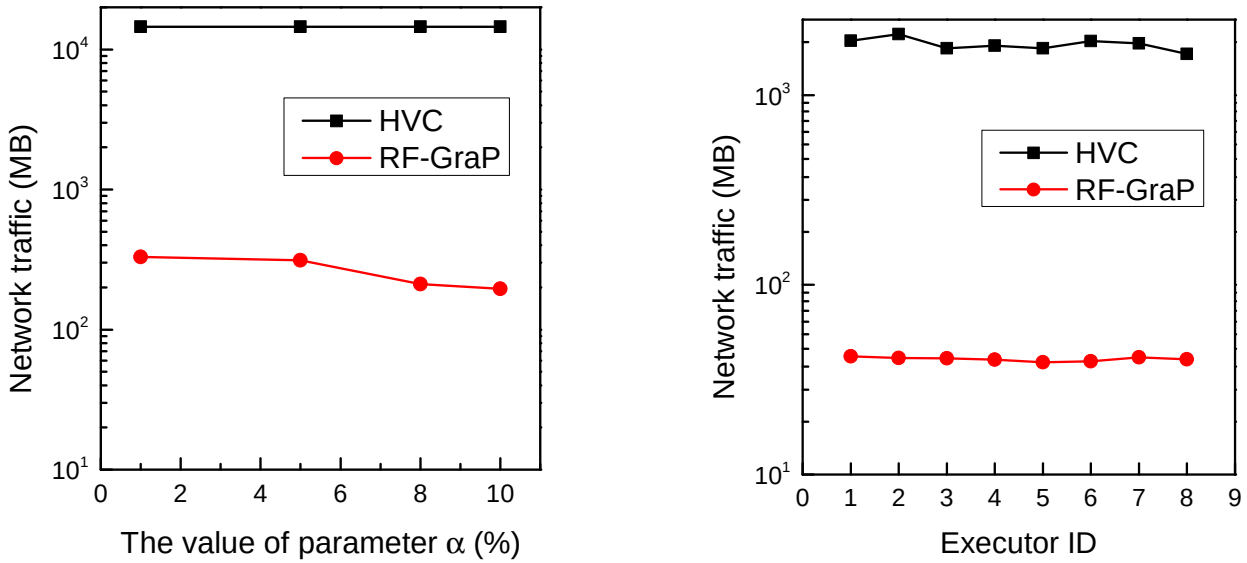
(b) varying the scale factor of number of attributes

Figure: Speedup achieved by RF-GraP over HVC with varying the scale factors over the default data (with 32 cores).

- RF-GraP can always achieve a significant speedup over the HVC method, and the speedup is becoming more obviously with increasing either the number of events or attributes

Performance Results - Network Communication

1. Communication cost is evaluated by recording the metric *Shuffle Read*.
2. Load balancing is evaluated by recording the *Shuffle Read* at each executor.



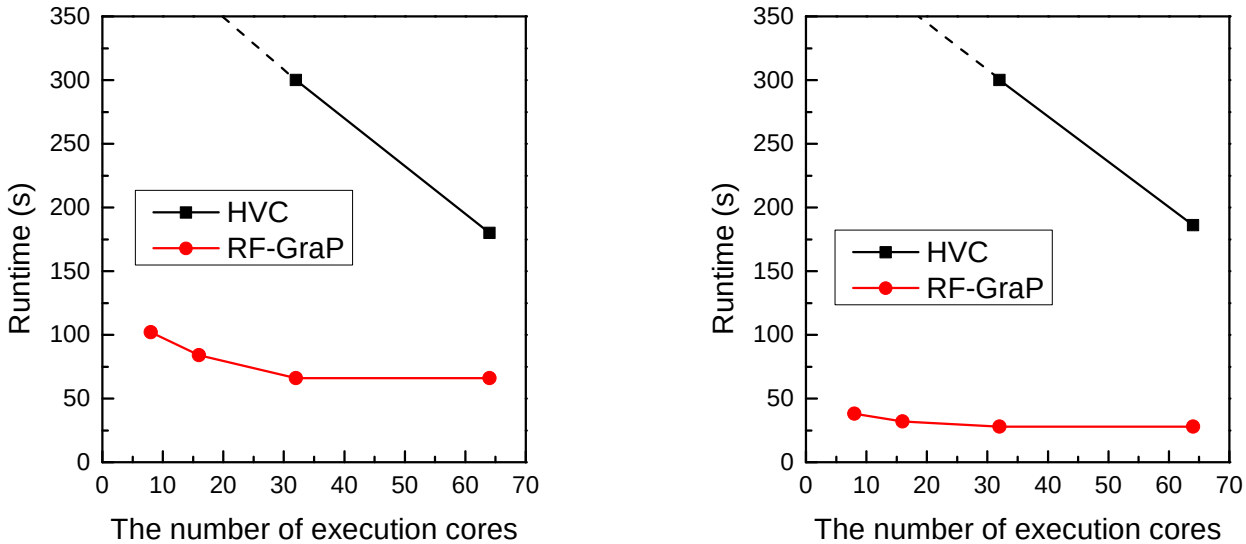
(a) general network traffic (b) retrieved data of each executor

Figure: The network communication of each algorithm when using 8 workers (log-scale).

- RF-GraP transfers about $43 - 74\times$ less data than HVC, with good loadbalancing.

Performance Results - Scalability

- Varying number of slaves (workers) over the default dataset, from 8 cores (2 nodes) to 64 cores.



(a) with $\alpha = 1\%$ (b) with $\alpha = 10\%$

Figure: Scalability of our algorithms by varying the number of executors.

- Benefit of adding more workers (i.e., the scaled speedup) decreases in RF-GraP: network and computing workloads are comparably small for the underlying platform.

Contributions

- We introduce the state-of-the-art event correlation approach and analyze its possible performance issues in processing large event logs over distributed systems.
- We adopt the principle of filtering-and-verification for efficient correlation analytics over large logs, by incorporating light-weight filter units into candidate correlation rules, which can be utilized to prune large numbers of non-interesting rules before verification.
- To further improve our performance, in the verification phase, we model existing correlation rules as a graph and introduce a graph partitioning approach to decompose the potentially correlated events into chunks by exploring efficient data locality assignment.
- We describe the implementation of our approach and report on experiments using Spark. The results demonstrate that we can achieve significant performance improvements over the state-of-the-art method. For example, for a log with 1 million events and 38 attributes, our algorithm performs 4.5$\times$ faster with 43$\times$ less network traffic.

Questions?