Towards an Extensible Efficient Event Processing Kernel

Mohammad Sadoghi*
Middleware Systems Research Group
Department of Computer Science
University of Toronto, Canada
mo@cs.toronto.edu

ABSTRACT

The efficient processing of large collections of patterns (Boolean expressions, XPath expressions, or continuous SQL queries) over data streams plays a central role in major data intensive applications ranging from user-centric processing and personalization to real-time data analysis. On the one hand, emerging user-centric applications, including computational advertising and selective information dissemination, demand determining and presenting to an end-user only the most relevant content that is both user-consumable and suitable for limited screen real estate of target (mobile) devices. We achieve these user-centric requirements through novel high-dimensional indexing structures and (parallel) algorithms. On the other hand, applications in real-time data analysis, including computational finance and intrusion detection, demand meeting stringent subsecond processing requirements and providing high-frequency and low-latency event processing over data streams. We achieve real-time data analysis requirements by leveraging reconfigurable hardware – FPGAs – to sustain line-rate processing by exploiting unprecedented degrees of parallelism and potential for pipelining, only available through custom-built, application-specific, and low-level logic design. Finally, we conduct a comprehensive evaluation to demonstrate the superiority of our proposed techniques in comparison with state-of-the-art algorithms designed for event processing.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Algorithms, Design, Measurement, Experimentation, Performance

Keywords

Boolean Expression Indexing, Publish/Subscribe, Complex Event Processing, Data Streams, Data-centric Workflows, and FPGAs

*PhD Adviser: Hans-Arno Jacobsen

1. INTRODUCTION

Efficient event processing is an integral part of a growing number of web and data management technologies ranging from user-centric processing and personalization to real-time data analysis. In user-centric processing applications, there are computational advertising [28, 10], online job sites [17, 28], and location-based services for emerging applications in the co-spaces [1, 19]; common to all are patterns and specifications (e.g., advertising campaigns, job profiles, service descriptions) modeled as Boolean expressions, XPath expressions, or SQL queries and incoming user information (e.g., user profiles and preferences) modeled as events using attribute-value pairs, XML document, or relational tuples. In the real-time analysis domain, there are (complex) event processing [11, 2, 6, 7, 5], XML filtering [3, 18, 15], intrusion detection [27], and computational finance [23]; again, common among these applications are predefined set of patterns (e.g., investment strategies and attack specifications) modeled as subscriptions and streams of incoming data (e.g., XML documents, data packets, stock feeds) modeled as events.

Unique to user-centric processing and personalization are strict requirements to determine only the most relevant content (e.g., ads) that is both user-consumable and suitable for the often limited screen real estate of client devices [17, 28, 10]. In addition, the user-centric processing demands scaling to millions of patterns and specifications (e.g., advertising campaigns) for supporting large-scale enterprise-level user-services, processing latency constraints in the subsecond range for meeting an acceptable service-level agreement, and improve expression expressiveness for capturing interesting patterns and desired preferences. To address these challenges using a software-based approach, we develop and design an effective algorithms and high-dimensional indexing structures [20, 8, 21, 22] to achieve processing large volume of incoming user information and to serve user-relevant contents.

Unique to real-time data analysis applications are critical requirements to meet the ever growing demands in processing large volumes of data at predictably low-latencies across many application scenarios. The need for more processing bandwidth is the key ingredient in high-throughput real-time data analysis that enables processing, analyzing, and extracting relevant information from streams of incoming data. Therefore, as proliferation of data and bandwidth continues, it is becoming essential to expand the research horizon to go beyond the conventional software-based approaches and adopt other key enabling technologies such as reconfigurable hardware in the form of FPGAs. FPGAs are cost-effective and energy-efficient solutions that are increasingly being explored to accelerate data management applications. On this front, using a hardware-based approach, through FPGAs, we exploit the inherent hardware parallelism by creating custom-built, application-specific, and low-level logic design in order to achieve real-time event processing on hardware [23, 26, 25, 24].

In particular, at the core of any efficient event processing platform lies a kernel that solves the matching problem (i.e., the focus of this thesis.) We define the stateless matching problem as:

Given an event ω (e.g., a user profile) and a set of Boolean expressions $\mathbf{\Omega}$ (e.g., advertising campaigns), find all expressions $\Omega_i \in \mathbf{\Omega}$ satisfied by ω.

This definition can be extended to support stateful matching by incorporating time- or count-based sliding window semantics in order to define the matching problem over a snapshot of an observed finite portion of the event stream. In addition, our expressions are reformulated as SQL queries (instead of Boolean expressions) extended with sliding window semantics. Likewise, the stateful matching problem is defined as follows:

Given a stream of events (i.e., a bag of $\langle \omega, \tau \rangle$ tuples, where ω is an event at time τ) and a collection of continuous SQL queries $\boldsymbol{Q}$, the SQL queries are continuously evaluated over the event stream.

We extract five challenges of paramount importance from applications that rely on event processing, in particular, when patterns are defined as Boolean expressions, which is the primary focus of this paper. First and foremost, the index structure designed for matching Boolean expressions must support top-k matching to quickly retrieve only the most relevant expressions, which is not addressed by prior Boolean expression matching approaches (e.g., [2, 7].) Furthermore, the relevance computation must be based on a generic and preference-aware scoring function, which is not addressed by prior approaches (e.g., [17].) Moreover, the top-k model must cope with much higher dimensionality (e.g., a dimension can represent an attribute in the user profile such as age, gender, and location) that is beyond the scope of dominant database top-k techniques (e.g., [14]) and skyline query processing (e.g., [4].) In general, the prevalent space dimensionality in event processing applications is in hundreds (or thousands), namely, orders of magnitude larger than capabilities of existing multi-dimensional and high-dimensional structures developed in the database community (e.g., [12].) Second, the index must support predicates with an expressive set of operators over continuous and discrete domains, which is also not supported by prior matching approaches (e.g., [2, 7, 28].) Third, the index must enable dynamic insertion and deletion of expressions, often disregarded as a requirement in matching algorithm design (e.g., [2, 17, 28].) Fourth, the index must employ a dynamic structure that adapts to changing workload distributions and expression schemata, also often disregarded in most matching algorithm designs (e.g., [2, 17, 28].) Finally, the index structure must scale to millions of Boolean expressions and afford low-latency and high-throughput expression matching demanded by most user-centric and real-time data analysis applications.

2. THESIS OUTLINE

This thesis addresses the problem of efficient event processing over data streams by proposing a solution that consists of three layers[1]: the higher-, the core-, and the lower-layers. At the core-layer of our solution lies an efficient Boolean expression *matching kernel* [20, 8, 21, 22]. The extensibility of this *matching kernel*[2] gives rise to the higher-layer of our solution that supports application requirements including top-k processing [22], XML filtering and dissemination [20], and (complex) event processing [8]. Another major benefit of our *matching kernel* is its regular, more tabular organization of its data structure, as well as Boolean expression and attribute-value pairs-based nature of its base language, that lends itself well to parallel and hardware processing. As a result, at the lower-layer, the event processing performance is further boosted by leveraging an unprecedented hardware parallelism [23, 26, 24, 25], only available through custom-built, application-specific, and low-level logic design. Figure 1 illustrates our three-layer architecture.

[1]As part of this thesis, we also developed a Boolean expression workload generator (BEGen): *http://msrg.org/datasets/BEGen*

[2]The matching kernel is implemented in C (47K lines of code.)

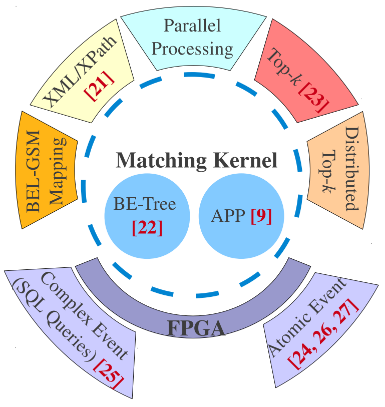

Figure 1: Thesis Overview

In particular, at the lower-layer, (1) we extend our solution to enable complex event processing [24] (i.e., stateful matching.) In particular, we develop high-throughput, custom circuits to implement the relational algebra over a window of input events for effective processing of SPJ (Select-Project-Join) queries [24]. The hardware implementation enables a high degree of parallelism and pipelining beyond the reach of software-based implementations. These custom circuits serve as a library of operators. (2) We introduce a novel multi-query optimization technique inspired from highly parallelizable rule-based system designs by mapping an SPJ-query into a Rete-like operator network [24]. We exploit the overlap among SPJ query plans by constructing a single global query plan to be executed in hardware. (3) We design software-to-hardware multi-query processing techniques that map a set of SPJ queries into a Rete-like global query plan. Subsequently, the global plan is converted into Hardware Description Language (HDL) code using our "hardware library" of custom building blocks for the various relational algebra operators. These mapping techniques are akin to a compiler that can process a query expressed in our language into a custom circuit that processes event streams.

At the higher-layer, we present BE*-Tree that is geared towards efficiently determining the most relevant expressions (e.g., ads), in which top-k processing is treated as first class citizens [22]. BE*-Tree introduces a novel hierarchical top-k processing scheme that differs from existing work which relies on a static and a flat structure [17, 28]. Furthermore, we present GPX-Matcher, a novel encoding of XPath expressions and XML documents into expressions and attribute-value pairs, respectively, that leverages our *matching kernel* [20]. Moreover, we formalize the encoding of XPath expressions into predicated-based Boolean expressions. We demonstrate a matching time in the millisecond range for millions of XPath expressions which outperforms state-of-the-art algorithms.

Finally, at the core-layer, the BE-Tree family structure achieves scalability by overcoming the curse of dimensionality through a (non-rigid) space-cutting technique without restricting the language expressiveness and structure dynamics [21, 22]. Notably, with respect to scalability, in BE*-Tree, we solve the two critical challenges common to most multi-dimensional and high-dimensional structures (e.g., [12]): (1) avoiding indexing non-empty space (2) minimizing overlap and coverage [22].

These challenges are tackled in BE*-Tree by proposing a bi-directional tree expansion: (1) a top-down (data and space clustering) and a bottom-up (space clustering) growths process, which together enable indexing only non-empty continuous subspaces and adapting to workload changes; (2) a splitting strategy to systematically produce and maintain overlap-free subspaces for holding expressions [22]. Furthermore, with respect to adaptability, in BE-Tree, we introduce a deterministic and a self-adjusting mechanisms that adapt as expression and event workloads change; thus, BE-Tree's main focus is to achieve an insertion-sequence independent dynamics [21].

In the following sections, we, first, formalize our Boolean expression language and data model; second, we briefly describe the (common) key structure of BE-Tree and BE*-Tree.

3. LANGUAGE AND DATA MODEL

In this section, we define our Boolean expression language and spatial event data model followed by our (top-k) matching semantics.

Notation Given an n-dimensional space $\Re^n$, we define the projection of $\Re^n$ onto $\Re^k$ as a k-dimensional subspace, denoted by $\pi_{d_1 \cdots d_k}(\Re^n) = \Re^k$, where $\pi_{d_1 \cdots d_k} : \Re^n \to \Re^k$, $k \leq n$, and each $d_i \in \{d_1 \cdots d_k\}$ represents the i^{th} dimension in $\Re^k$ and corresponds to the j^{th} dimension in $\Re^n$; for ease of notation, we define the identity projection as $\pi_{\mathcal{I}}(\Re^n) = \Re^n$. In addition, we define a k-dimensional bounding box $\mathcal{B}^k$ over $\Re^k$ as

$$\mathcal{B}^k = [\mathtt{min}_1, \mathtt{max}_1] \times \cdots \times [\mathtt{min}_k, \mathtt{max}_k].$$

Let $\xi_j(\mathcal{B}^k) = [\mathtt{min}_i, \mathtt{max}_i]$ be the i^{th} boundary in $\mathcal{B}^k$ defined over the i^{th} dimension in $\Re^k$ which corresponds to the j^{th} dimension in $\Re^n$. Let $\chi_i(\mathcal{B}^k)$ be the center of the i^{th} boundary in $\mathcal{B}^k$. Furthermore, let $\lambda_i(\mathcal{B}^k)$ be the length of the i^{th} boundary in $\mathcal{B}^k$ given by

$$\lambda_i(\mathcal{B}^k) = \mathtt{max}_i - \mathtt{min}_i.$$

Lastly, let $\mu_i(\mathcal{B}^k) = \mathtt{min}_i$ and $M_i(\mathcal{B}^k) = \mathtt{max}_i$ represent the minimum and the maximum value of the i^{th} boundary of $\mathcal{B}^k$, respectively.

Expression Language and Subspace Model We support a rich Boolean expression language that unifies the subscription language and the event data model. This generalization gives rise to more expressive matching semantics while still encompassing the traditional publish/subscribe matching problem.

In our model, a Boolean expression is a conjunction of Boolean predicates. A predicate is a quadruple: an attribute that uniquely represents a dimension in $\Re^n$; an operator (e.g., relational operators ($<, \leq, =, \neq, \geq, >$), set operators ($\in, \notin$), and the SQL BETWEEN operator); a set of values (for discrete domains) or a range of values (for continuous domains); and an assigned predicate weight, denoted by $P^{\mathtt{attr,opt,val,wt}}(x)$ or more concisely as $P(x)$. A predicate either accepts or rejects an input x such that $P^{\mathtt{attr,opt,val,wt}}(x) : x \to \{\mathtt{True}, \mathtt{False}\}$, where $x \in \mathtt{Dom}(P^{\mathtt{attr}})$ and $P^{\mathtt{attr}}$ is the predicate's attribute. Formally, a Boolean expression Ω is defined over $\Re^n$ as follows:

$$\Omega = \{P_1^{\mathtt{attr,opt,val,wt}}(x) \wedge \cdots \wedge P_k^{\mathtt{attr,opt,val,wt}}(x)\},$$
$$where\ k \leq n;\ \ i, j \leq k,\ P_i^{\mathtt{attr}} = P_j^{\mathtt{attr}}\ \ iff\ \ i = j.$$

We extended the semantics of projection to a Boolean expression in order to enable projecting onto predicates associated with certain dimensions

$$\pi_{d_1 \cdots d_h}(\Omega) = \{\Omega' \mid \Omega' = P_1^{\mathtt{attr,opt,val,wt}}(x) \wedge \cdots \wedge P_h^{\mathtt{attr,opt,val,wt}}(x),$$
$$\forall d_i \in \{d_1 \cdots d_h\},\ P_i^{\mathtt{attr}} = d_i,\ P_i(x) \in \Omega\}.$$

Table 1: Predicate Mapping

$P^{\mathtt{attr,opt,val,wt}}(x)$	$\gamma(P^{\mathtt{attr,opt,val,wt}}(x))$
$\mathtt{attr} < v_1$	$[-\infty, v_1 - \epsilon]$
$\mathtt{attr} \leq v_1$	$[-\infty, v_1]$
$\mathtt{attr} = v_1$	$[v_1, v_1]$
$\mathtt{attr} \neq v_1$	$[-\infty, \infty]$
$\mathtt{attr} \geq v_1$	$[v_1, \infty]$
$\mathtt{attr} > v_1$	$[v_1 + \epsilon, \infty]$
$\mathtt{attr} \in \{v_1, \cdots, v_m\}$	$[v_1, v_m]$
$\mathtt{attr} \notin \{v_1, \cdots, v_m\}$	$[-\infty, \infty]$
$\mathtt{attr}$ BETWEEN v_1, v_2	$[v_1, v_2]$

Now, we are in a position to formalize the predicate mapping[3] and, ultimately, to (approximately) represent an expression as a k-dimensional bounding box.

A predicate $P^{\mathtt{attr,opt,val,wt}}(x)$ is mapped into a 1-dimensional bounding box, denoted by $\gamma(P^{\mathtt{attr,opt,val,wt}}(x)) = \mathcal{B}^1$, as shown in Table 1, where ϵ is the machine's epsilon. Therefore, a predicate $P(x)$ is covered by $\mathcal{B}^1 = [\mathtt{min}_1, \mathtt{max}_1]$ only if the permitted values defined by $P(x)$ lie in $[\mathtt{min}_1, \mathtt{max}_1]$; for brevity, we say the predicate is enclosed by $\mathcal{B}^1$. Similarly, we say an expression Ω over $\Re^n$ is partially enclosed by $\mathcal{B}^h$ w.r.t. the projection $\pi_{d_1 \cdots d_h}$, denoted by $\Gamma^{\pi}(\Omega)$.

$$\Gamma^{\pi}(\Omega) = \{\mathcal{B}^h \mid \forall P_i^{\mathtt{attr,opt,val,wt}}(x) \in \pi(\Omega),$$
$$\gamma(P_i(x)) \cap \xi_{P_i^{\mathtt{attr}}}(\mathcal{B}^h) = \gamma(P_i(x))\}.$$

Furthermore, we say an expression Ω is fully enclosed by $\mathcal{B}^k$ when the identity projection $\pi_{\mathcal{I}}$ is applied, $\Gamma^{\pi_{\mathcal{I}}}(\Omega) = \mathcal{B}^k$. Lastly, the smallest $\mathcal{B}^k$, or the minimum bounding box (MBB), that (partially) encloses an expression Ω is given by

$$\Gamma^{\pi}_{\mathtt{min}}(\Omega) = \{\arg\min_{\mathcal{B}_i^k}\ \ \Gamma^{\pi}(\Omega) = \mathcal{B}_i^k\}.$$

The Boolean expression Ω is said to have size k, denoted by $|\Omega| = k$, when having k predicates; hence, Ω is represented by $\Gamma_{\mathtt{min}}(\Omega)$ defined over a k-dimensional subspace.

Top-k Matching Semantics Our formulation of subscriptions and events as expressions enables us to support a wide range of matching semantics, including the classical publish/subscribe matching: *Given an event ω and a set of subscriptions $\mathbf{\Omega}$, find all subscriptions $\Omega_i \in \mathbf{\Omega}$ that are satisfied by ω.* We refer to this problem as the *stabbing subscription*[4] $\mathtt{SQ}(\omega)$ and formalize it as follows:

$$\mathtt{SQ}(\omega) = \{\Omega_i \mid \forall P_q^{\mathtt{attr,opt,val,wt}}(x) \in \Omega_i,\ \exists P_o^{\mathtt{attr,opt,val,wt}}(x) \in \omega,$$
$$P_q^{\mathtt{attr}} = P_o^{\mathtt{attr}}, \exists x \in \mathtt{Dom}(P_q^{\mathtt{attr}}),\ P_q(x) \wedge P_o(x)\}.$$

Alternatively, we can (approximately) express *stabbing subscription* as subspace matching in $\Re^n$ as follows, where the approximation is due to the mapping function γ:

$$\mathtt{SQ}(\omega) = \{\Omega_i \mid \forall P_q^{\mathtt{attr,opt,val,wt}}(x) \in \Omega_i,\ \exists P_o^{\mathtt{attr,opt,val,wt}}(x) \in \omega,$$
$$P_q^{\mathtt{attr}} = P_o^{\mathtt{attr}},\ \gamma(P_q(x)) \cap \gamma(P_o(x)) \neq \emptyset\}.$$

We also adopt the popular vector space scoring used in information retrieval (IR) systems[5] for computing the score of a matched subscription Ω_i for a given event ω (to enable top-k computation), denoted by $\mathtt{score}(\omega, \Omega_i)$, and defined by

[3]The mapping strategy for predicates with operator $\in, \notin, \neq$ is especially effective because the number of predicates per expression is on the order of tens while the number of space dimensions is on the order of thousands.

[4]This is a relaxation of the stabbing query problem, in which interval cutting is generalized to subspace cutting.

[5]We are not limited to IR scoring, but support any monotonic scoring function.

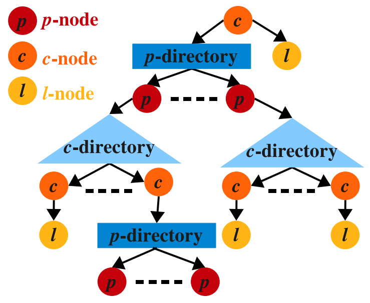

Figure 2: An overview of BE-Tree/BE*-Tree data structure.

$$\texttt{score}(\omega, \Omega_i) = \sum_{P_q(x)\in\Omega_i, P_o(x)\in\omega, P_q^{\texttt{attr}}=P_o^{\texttt{attr}}} P_q^{\texttt{wt}} \times P_o^{\texttt{wt}}.$$

Similarly, we compute the upper bound score for an event ω w.r.t. an upper bound weight-summary ($\texttt{sum}_{\texttt{wt}}^{\boldsymbol{\Omega}}$) for a set of subscriptions $\boldsymbol{\Omega}$ as follows

$$\texttt{uscore}(\omega, \texttt{sum}_{\texttt{wt}}^{\boldsymbol{\Omega}}) = \sum_{P_o(x)\in\omega} P_o^{\texttt{wt}} \times \texttt{sum}_{\texttt{wt}}^{\boldsymbol{\Omega}}(P_o^{\texttt{attr}}),$$

where $\texttt{sum}_{\texttt{wt}}^{\boldsymbol{\Omega}}(\texttt{attr})$ returns the upper bound score of $\texttt{attr}$ over the set of subscriptions $\boldsymbol{\Omega}$ which is given by

$$\texttt{sum}_{\texttt{wt}}^{\boldsymbol{\Omega}}(\texttt{attr}) = \max_{\Omega_i\in\boldsymbol{\Omega}, P_q(x)\in\Omega_i, P_q^{\texttt{attr}}=\texttt{attr}} P_q^{\texttt{wt}}.$$

4. BOOLEAN EXPRESSION INDEXING

As part of this thesis, we develop two novel data structures, namely, BE-Tree [21] and BE*-Tree [22], which are generic index structures for indexing a large collection of Boolean expressions (i.e., subscriptions) and for efficient retrieval of the most relevant matching expressions given a stream of incoming events.

BE-Tree/BE*-Tree Overview BE-Tree (and BE*-Tree[6]) supports Boolean expressions with an expressive set of operators defined over a high-dimensional continuous space. BE-Tree copes with the curse of dimensionality challenge through a (non-rigid) two-phase space-cutting technique that significantly reduces the complexity and the level of uncertainty of choosing an effective criterion to recursively cut the space and that identifies highly dense subspaces. The two phases BE-Tree employs are: (1) *partitioning* which is the global structuring to determine the next best attribute $\texttt{attr}_i$ (i.e., the i^{th} dimension in $\Re^n$) for splitting the space and (2) *(non-rigid) clustering* which is the local structuring for each partition to determine the best grouping of expressions w.r.t. the expressions' range of values for $\texttt{attr}_i$. In addition, BE*-Tree not only supports dynamic insertion and deletion of expressions similar to BE-Tree, but it also adapts to workload distributions by incorporating top-down (data and space clustering) and bottom-up (space clustering) expansion within each clustering phase.

The data clustering aims to avoid indexing empty space and to adapt to a skewed workload while space clustering aims to avoid degeneration of the structure in the presence of frequent insertions and deletions of expressions. Conceptually, BE*-Tree's space and data clustering techniques are a hybrid scheme that takes the best of both worlds. On the one hand, inspired by adaptiveness of R-Tree based structure, the data clustering employs data dependent grouping of expressions to adapt to different workload distributions and to avoid indexing empty space, and, on the other hand, inspired by robustness of grid-based structure, the space clustering employs space dependent grouping of expressions to accommodate an insertion-independent mechanism. One of the key distinguishing factor between BE-Tree and BE*-Tree is the above-mentioned non-rigid clustering structure proposed in BE*-Tree, which due to limited space is not further discussed.

[6]The basic structure of BE-Tree and BE*-Tree are similar; thus, what follows applies equally to BE*-Tree as well.

In general, BE-Tree is an n-ary tree structure in which a leaf node contains the actual data (expressions) and an internal node contains partial information about data (e.g., an attribute and a range of values) in its descendant leaf nodes. BE-Tree consists of three classes of nodes: p-node (partition node) for storing the partitioning information, i.e., an attribute; c-node (cluster node) for storing the clustering information, i.e., a range of values; and l-node (leaf node), being at the lowest level of the tree, for storing the actual data. Moreover, p-nodes and c-nodes are logically organized in a special directory structure for fast tree traversal and search space pruning. Thus, a set of p-nodes are organized in a p-directory (partition directory), and a set of c-nodes are organized in a p-directory (cluster directory.) The overall BE-Tree structure is depicted in Figure 2.

Two-phase Space-cutting BE-Tree's two-phase space-cutting, the partitioning followed by the clustering, introduces new challenges such as how to determine the right balance between the partitioning and clustering, and how to develop a robust principle to alternate between both. BE-Tree partially addresses these challenges by exploiting the discrete and finite space and by relying on deterministic grid-based clustering. Thus, in BE-Tree, we propose a splitting policy to guide the clustering phase for establishing not only a robust principle for alternating between partitioning and clustering but also for naturally adapting to the workload distributions. We begin discussing the overall structure and the two-phase cutting dynamics of BE-Tree before presenting the key design principles behind BE-Tree.

In BE-Tree, the partitioning phase, conceptually a global adjusting mechanism, is the first phase of our space-cutting technique. The partitioning phase is invoked once a leaf node overflows (initially occurs at the root level.) This phase involves ranking each candidate $\texttt{attr}_i$, attributes that appear in the expressions of the overflowed l-node, in order to determine the most effective $\texttt{attr}_i$ for partitioning. Essentially, this process identifies the highest ranking $\texttt{attr}_i$, given a scoring function, to spread expressions into smaller groups (leaf nodes.) In short, partitioning space based on a high-ranking $\texttt{attr}_i$ enables the pruning of search space more efficiently while coping with the curse of dimensionality by considering a single $\texttt{attr}_i$ for each partitioning phase.

Upon executing the partitioning phase, the high-dimensional indexing problem is reduced to one-dimensional interval indexing, which paves the way to exploit underlying distribution of a single $\texttt{attr}_i$ at a time through BE-Tree's clustering phase, conceptually a local adjusting mechanism. The clustering phase, and ultimately the key component of BE-Tree, consists of (1) a clustering policy to group overlapping expressions (into buckets) that minimizes the overlap and the coverage among these buckets and (2) a robust and well-defined policy to alternate between the partitioning and the clustering.

The absence of a well-defined policy gives rise to the dilemma of whether to further pursue the space clustering or to switch back to the partitioning. Furthermore, a partitioned bucket can no longer be split without suffering from the cascading split problem. Thus, a clustering policy that cannot react and adapt to the insertion sequence is either prone to ineffective buckets that do not take advantage of the domain selectivity for effectively pruning the search space or is prone to suffering from substantial performance overhead due to the cascading split problem. Therefore, a practical space clustering must support dynamic insertion and deletion and must adapt to any workload distributions, yet satisfying the cascading-split-free property.

In the clustering phase, each group of expressions is referred to as a bucket. Formally, a *bucket* is a 1-dimensional bounding box over $\mathtt{attr}_i$, denoted by $\mathcal{B}^1$, and an expression Ω with predicate $P_j^{\mathtt{attr},\mathtt{opt},\mathtt{val},\mathtt{wt}}(x) \in \Omega$ is assigned to a bucket only if

$$\mathtt{attr}_i = P_j^{\mathtt{attr}} \text{ and } \gamma(P_j(x)) \cap \xi_{P_j^{\mathtt{attr}}}(\mathcal{B}^1) = \gamma(P_j(x)).$$

Furthermore, a bucket has a minimum bounding box (MBB) that partially encloses all of its expressions $\mathbf{\Omega}$ if and only if

$$\mathcal{B}^1 = \bigcup_{\Omega_j \in \mathbf{\Omega}} \Gamma_{\mathtt{min}}^{\pi_{\mathtt{attr}_i}}(\Omega_j).$$

Moreover, each bucket is associated with exactly one c-node in the BE-Tree, which is responsible for storing and maintaining information about the bucket's assigned expressions.

BE-Tree/BE*-Tree Invariance In order to formalize BE-Tree's invariance and operational semantics, we must distinguish among four bucket types: *open bucket*: a bucket that is not yet partitioned; *leaf bucket*: a bucket that has no children (or has not been split); *atomic bucket*: a bucket that is a single-valued bucket which cannot further be split; and *home bucket*: a bucket that is the smallest existing bucket that encloses the inserting expression.

The correctness of the BE-Tree operational semantics is achieved based on the following three rules:

1. *insertion rule*: an expression is always inserted into the smallest bucket that encloses it.
2. *forced split rule*: an overflowing non-leaf bucket is always split before switching back to the partitioning.
3. *merge rule*: an underflowing leaf bucket is merged with its parent only if the parent is an open bucket

Finally, the BE-Tree correctness can be summarized as follows. INVARIANCE: *Every expression* Ω *is always inserted into the smallest bucket that encloses it and a (non-atomic) bucket is always split first before it is partitioned.*

5. FUTURE THESIS DIRECTIONS

At the lower-layer of this thesis proposal, we continue to experiment with various designs that further exploit hardware-parallelism; in particular, we are currently evaluating a new hardware encoding to utilize the fast on-chip registers and a data placement algorithm (through replication and horizontal data partitioning) that eliminates chip's idle areas and resources.

At the higher- and core-layers, in order to enrich expression expressiveness, we aim to support arbitrary Boolean expressions that are not limited to only disjunctive normal form expression; a requirement demanded in many application scenarios such as computational advertising and real-time data analysis. More importantly, in order to handle the sheer volume of today's social and enterprise data, we are extending our algorithm to enable parallel matching (using OpenMP) and distributed top-k processing and exploring the possibility of matching computation distribution over MapReduce abstraction model.

Lastly, at the higher-layer, in order to study the generality, effectiveness, and practicality of our matching semantics and event processing language and data model expressiveness, we are developing safe distribution and parallel execution of data-centric workflows over our event processing kernel (i.e., a key building block in the publish/subscribe paradigm.) In essence, we are presenting a novel reformulation of data-centric workflow that is designed to utilize the loosely decoupled and distributed nature of publish/subscribe systems. Furthermore, we are demonstrating the practicality and expressiveness of our proposal by mapping an industry-based data-centric workflow, namely, IBM business artifact with Guard-Stage-Milestone (GSM), into our (extended) event processing language and data model. In short, the contributions are (1) mapping of data-centric workflow into publish/subscribe abstraction to achieve distributed and parallel execution; (2) detailed theoretical analysis of our proposed mapping; (3) formalizing the complexity of optimal workflow distribution over the publish/subscribe paradigm; and (4) implementing our proposed mapping over PADRES [9, 16], an enterprise-grade publish/subscribe infrastructure.

6. REFERENCES

[1] R. Agrawal, A. Ailamaki, P. A. Bernstein, et al. The claremont report on database research. *SIGMOD Rec.'08*.

[2] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC'99*.

[3] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB'00*.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE'01*.

[5] L. Brenna, A. Demers, J. Gehrke, M. Hong, Ossher, Panda, Riedewald, Thatte, and White. Cayuga: high-performance event processing engine. *SIGMOD'07*.

[6] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith. Efficient filtering in publish-subscribe systems using binary decision diagrams. In *ICSE'01*.

[7] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for fast pub/sub systems. *SIGMOD'01*.

[8] A. Farroukh, M. Sadoghi, and H.-A. Jacobsen. Towards vulnerability-based intrusion detection with event processing. In *DEBS'11*.

[9] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski. The padres distributed publish/subscribe system. In *ICFI'05*.

[10] M. Fontoura, S. Sadanandan, J. Shanmugasundaram, S. Vassilvitski, E. Vee, S. Venkatesan, and J. Zien. Efficiently evaluating complex Boolean expressions. In *SIGMOD'10*.

[11] C. Forgy. Rete: A fast algorithm for the many patterns/many objects match problem. *Artif. Intell.'82*.

[12] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.'98*.

[13] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *WS-FM'10*.

[14] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.'08*.

[15] G. Li, S. Hou, and H.-A. Jacobsen. Routing of XML and XPath queries in data dissemination networks. In *ICDCS'08*.

[16] G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM TWEB'10*.

[17] A. Machanavajjhala, E. Vee, M. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. *VLDB'08*.

[18] M. M. Moro, P. Bakalov, and V. J. Tsotras. Early profile pruning on XML-aware publish-subscribe systems. In *VLDB'07*.

[19] B. Ooi, K. Tan, and A. Tung. Sense the physical, walk through the virtual, manage the co (existing) spaces: A database perspective. In *SIGMOD Rec.'09*.

[20] M. Sadoghi, I. Burcea, and H.-A. Jacobsen. GPX-Matcher: a generic Boolean predicate-based XPath expression matcher. In *EDBT'11*.

[21] M. Sadoghi and H.-A. Jacobsen. BE-Tree: An index structure to efficiently match Boolean expressions over high-dimensional discrete space. In *SIGMOD'11*.

[22] M. Sadoghi and H.-A. Jacobsen. Relevance matters: Capitalize on less (top-k matching in publish/subscribe). In *ICDE'12*.

[23] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H.-A. Jacobsen. Efficient event processing through reconfigurable hardware for algorithmic trading. In *VLDB '10*.

[24] M. Sadoghi, R. Javed, N. Tarafdar, H. Singh, R. Palaniappan, and H.-A. Jacobsen. Multi-query stream processing on fpgas. In *ICDE'12*.

[25] M. Sadoghi, H. Singh, and H.-A. Jacobsen. fpga-ToPSS: Line-speed event processing on fpgas. In *DEBS'11*.

[26] M. Sadoghi, H. Singh, and H.-A. Jacobsen. Towards highly parallel event processing through reconfigurable hardware. In *DaMoN'11 (Collocated with SIGMOD)*.

[27] D. Srivastava, L. Golab, R. Greer, T. Johnson, J. Seidel, V. Shkapenyuk, O. Spatscheck, and J. Yates. Enabling real time data analysis. *PVLDB'10*.

[28] S. Whang, C. Brower, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, R. Yerneni, and H. Garcia-Molina. Indexing Boolean expressions. In *VLDB'09*.