Stratified implementation of event processing network

Ayelet Biger
One Software Technologies
ayeletb@one1.co.il

Opher Etzion
IBM Haifa
Research Lab
opher@il.ibm.com

Yuri Rabinovich
IBM Haifa
Research Lab
yurir@il.ibm.com

ABSTRACT
This abstract describes an approach for stratified implementation of event processing network; the abstract explains the motivation, outlines the reasoning behind this approach and provides initial experimentation.

General Terms
Performance, Design, Experimentation.

Keywords
Event Processing, Stratification, Parallel computing.

1. INTRODUCTION
Most implementation of event processing applications are done by centralized engines, one of the main reasons is the inter-dependencies among various event processing operations, by the fact that an event processing operation may consume a derived event that has been created by another event processing operation, and implementations are built around the fact that these dependencies are resolved internally in the event processing engine.

EPN (Event Processing Networks) [1], [2] is a means to describe and implement event processing, under this paradigm the events are flowing among autonomic EPAs (Event Processing Agents); each EPA is performing a single operation. The EPN concept can model event processing applications, however it does neither describe the level of distribution nor the level of parallelism among agents, thus an EPN can be implemented anywhere between the two extremes of having an EPN describing a traditional centralized single engine implementation, and fully distributed environment in which every agent resides on a different machine. The goal of this research project is to bridge the gap between the model and implementation and optimize the configuration of allocation of agents to software artifacts and physical machines.

2. STRATIFICATION
The first step towards optimal implementation of EPN has been to apply the stratification approach, according to which all event processing operations are mapped to agents, and analyzed for dependencies and creates a dependency graph [3]; the dependency graph is then collapsed to a collection of strata, each stratum contains all agents that have any dependency in events that are derived within the previous stratum. Formally speaking:

- Let α be an EPA, such that their inputs are events αe_1 ,… αe_n..
- Let emitter (e) = source of event e; where emitter can be either external producer for raw events or an EPA for derived event
- Let emitter-stratum (e) = 0 if e is a raw event; stratum (emitter (e)) otherwise.
- α is assigned to stratum (i+1) iff
  max (emitter-stratum (αe_j) = i.

This recursive assignment rule partitions all agents to N strata level, according to the semantics of the dependencies among agents. Stratification by itself may not provide the optimal partition, but it may provide a basis for further optimizations, using the simplest assignment, of assign each stratum to a single processor, provides a first level of optimization. It should be noted that stratification has been used before in the active database area for modularization purposes [4]

3. PRAGMATIC CHALLENGES
Trying to apply the stratification approach to an event processing implementation whose basic programming model follows the EPN principles is straightforward. However, implementation using event processing model that hides the dependency inside an engine's implementation may be a challenge, depending on the level of meta-data that exists within the agent. Our first prototype is based on AMiT [5] which does not expose a dependency model, furthermore, there are various types of dependencies in events; e.g. AMiT supports context-oriented operations [6] which create extra dependencies in emitters of events that open or close contexts. Example: an agent traces complain on pain within the first 24 hours of a medical treatment, the dependency is not only in the event of "pain complain", but also in the event of "medical treatment occurred", which opens the temporal context. Other event processing languages may have other types of dependencies.

4. IMPLEMENTATION EXAMPLE

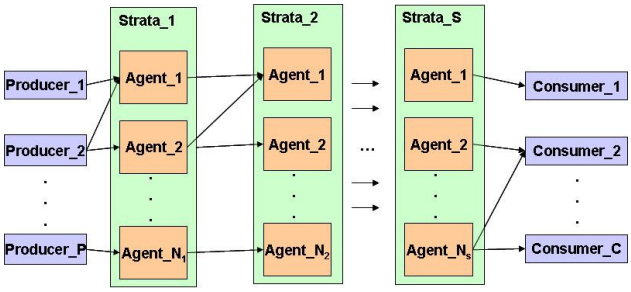

Figure 1: Stratified Architecture

Figure 1 shows an implementation architecture in which each stratum is implemented using different machines. There are P event producers and C event consumers. Each stratum level *i* may have N_i event processing agents and each agent handles single pattern. Each stratum level may run on different machine or multiple machines.

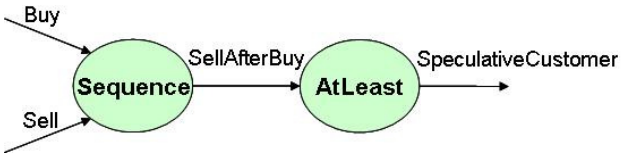

Figure 2: A simple example

Figure 2 describes a simple example that detects "speculative customers". It gets events about buy and sell of securities, its first agent detects customers that have bought and then sold the same security in the same day in a value of more than $1M, while the second agent determines that if a customer satisfied the first pattern at least three times within a month, it shows speculative behavior.

5. EXPERIENCE RESULTS

Implementation	Input events total throughput	Derived events total throughput
Single physical node (100%)	28,402	3,155
2 Stratum levels – 4 physical nodes. 3 agents in first stratum and 1 agent in second stratum	3x 27,292 + 31,092 = 112,968	10,364
4 nodes improvement	397.7%	328.5%
Additional node improvement	99.42%	82.12%

Table 1: Comparison of one and two levels

Table 1 shows measurements of event throughput for a centralized implementation with one physical processing node vs. the two strata implementation that uses 4 physical processing nodes, each running only one agent. The Sequence agent produces derived events in a rate that is 1/3 of the event input capacity of the AtLeast agent, thus we can use 3 agents in the first stratum to achieve max throughput of the stratified application. While there is a communication overhead, it shows throughput performance improvement of 397.7%. While throughput is just one of the optimization criteria, and the example is rather simple, this exercise served as a proof of concept for the usability of this approach. The continuation of this research will investigate:

1. Tradeoffs in further parallelization within a single stratum.
2. Optimization relative to other criteria of scalability, e.g. Number of producers, number of consumers, number of agents, size of state.
3. Taking location constraints into account for distribution of the system.
4. Taking into account heterogeneity of agent types.

6. References

[1] L. Perrochon, W. Mann, S.e Kasriel and D. Luckham (1999): Event Mining with Event Processing Networks. PAKDD 1999: 474-478

[2]. G. Sharon, O. Etzion: Event-processing network model and implementation (2008). IBM System Journal 47(2): 321-334

[3]. A. Biger - Complex Event Processing Scalability by Partition (2007) – M.Sc. Thesis, Technion – Israel Institute of Technology.

[4]. El. Baralis, S. Ceri, S. Paraboschi (1996): Modularization Techniques for Active Rules Design. ACM Trans. Database Syst. 21(1): 1-29

[5]. A. Adi, O. Etzion: Amit - the situation manager. VLDB J. 13(2): 177-203 (2004)

[6]. A. Adi, A. Biger, D. Botzer, O. Etzion, Z Sommer (2003): Context Awareness in Amit. Active Middleware Services 2003: 160-167