Distribution and Coordination of Policies for Large-scale Service Management

Nigel Sheridan-Smith[1], Tim O'Neill[1], John Leaney[1], and Mark Hunter[2]

[1] Institute of Information and Communication Technologies
University of Technology, Sydney
{nigelss, toneill, jrleaney}@eng.uts.edu.au
[2] Alcatel Australia
Mark.Hunter@alcatel.com.au

Abstract. The distribution and coordination of policies is often overlooked but is crucial to the scalability of dynamic, personalised services. In this work we partition an *Abstract Syntax Tree* of the policies to determine the responsibility of different management nodes in a geographically segregated network (i.e. *management by delegation*). This partitioning is combined with IN/OUT set analysis to determine the required coordination for policy enforcement of complex policies with inter-dependencies. Our simulation results show that this approach is promising, as higher decision loads can be readily handled by further sub-dividing of the network.

1 Introduction

The *PRONTO* management system [1] which is currently in development uses a policy-based service definition language to describe how to construct and manage personalised services in Next-Generation Networks (NGNs). Centralised policy architectures are unlikely to cope with millions of dynamically changing services and the Policy Decision Points (PDPs) will quickly become bottlenecks. Since *PRONTO* is evolutive, only one PDP is required for different types of policies as the network architecture evolves. Policies are distributed to multiple management nodes which are given responsibility for geographic partitions of the network. This approach allows policy load to be quickly redistributed by changing the demarcation of responsibility.

Whilst some types of policies can be easily distributed to multiple management nodes (due to their independence), other types of policies will require coordination because the events, conditions and actions (ECA) refer to devices in different network partitions. We apply the principles of *automatic parallelising compilers* [2] to aid in distributing and coordinating more complex policies where the sequencing of ECA components is critical to correct policy evaluation and enforcement. Further work in [4] examines many of the proposed extensions (flow control, looping and transactions) to the basic scheme outlined in this paper.

2 Policy distribution

2.1 The *PRONTO* management system and language

The *PRONTO* management system is highly flexible and extensible, allowing policies to be applied to interchangeable, event-driven software components, and devices from different vendors for end-to-end service management. The *service definition language* allows the behaviour of complex, adaptive and dynamic services to be described at multiple levels of abstraction, and allows the services to evolve over time. The language ties together the different aspects of the service, such as the devices involved (through *roles*), the parameters of the service, the software components used, the resources required, the different states of the service (for *workflow*), and the policies which define the static and dynamic behaviour of the services. Policies applied at abstract layers on software components create more detailed policies at lower layers, but the policies can be changed dynamically in response to feedback from the network, to optimise the configuration or adapt services to the user's environment. This approach is generic and elastic, therefore lending itself to the management of many different types of policy functions in the network (e.g. QoS, security, VPNs, multicast, etc), without multiple policy system implementations. For further details refer to [1].

2.2 Parallelisation of policies

In automatic parallelising compilers, the design of effective partitioning algorithms is difficult because coordination overhead trades off against parallelisation [2]. However, policy systems might benefit from *fine-grained* partitioning because the policy targets are distributed throughout the whole network with latency having only marginal impact.

We introduce three types of coordination: (1) *Data Distributions* (DI) where data dependencies exist across geographical partitions and shared data must be transmitted between the management nodes (2) *Sequence Points* (SP) where control flow sequencing occurs across geographical partitions, and (3) *Event Notifications* (EN) where the combination of events and conditions leads to policy actions being triggered across the network (all participating management nodes must be informed of this occurrence). Similar coordination can be applied equally to events and conditions, but only actions are shown in this paper. We distribute event management similar to Howard et al. [5] to minimise polling and communication overheads.

The **concurrent** and **sequential** keywords are used to allows policy writers to specify the optimal placement of parallelisation, and indicate where strict sequential execution must be observed. Concurrent actions can be executed separately and simultaneously as long as there are no dependencies; however *flow* or *data dependencies* must be ordered correctly when executed [2, 3]. Other types of interactions are *anti-dependencies* and *output dependencies*, and are due to the potential for race conditions.

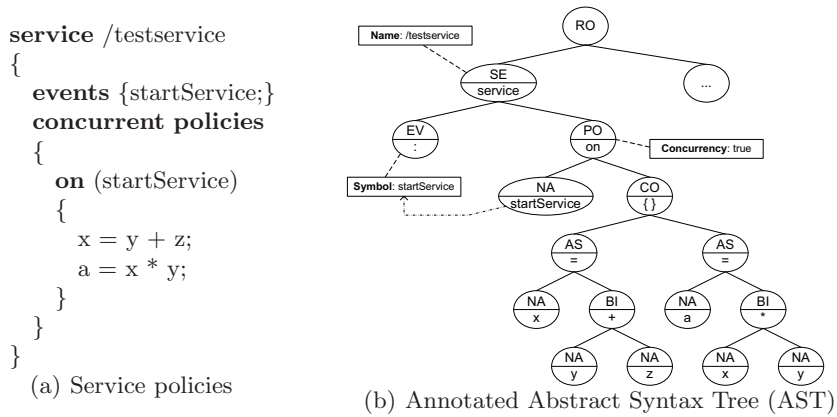


Fig. 1. Service to AST transformation

3 The distribution and coordination algorithm

In the simplest cases, a policy might be entirely managed by a single management node, or a purely concurrent policy might be distributed to independent management nodes with no coordination. In these cases, only an EN message is required to share event instance information and begin policy execution.

3.1 Abstract Syntax Tree partitioning

During policy compilation, an *Abstract Syntax Tree* (AST) is constructed, and context-sensitive information is *annotated* to the nodes [2], as shown in Fig. 1. The top half of each ellipse node is marked with the type of node and the bottom half is marked with the relevant text.

We can partition the different branches, and then associate the different branches with particular management nodes. The most important node is the *name expression* (NA) since it refer to various symbols, such as variables and the references to objects under management. Using two-phase depth-first traversals we mark on the AST nodes with arrows indicating the direction of responsibility. The NA nodes are directly associated with a management node that is responsible for the identified symbol (e.g. a device). In a downward pass, nodes below NA are marked upwards. In the upward pass, any unmarked nodes are marked downwards if all children belong to the same management node. These arrows then indicate regions of responsibility. In the second pass, DI nodes are inserted into the AST to clearly define the boundaries between partitions.

3.2 Data dependencies

The previous approach identifies simple dependencies within individual actions, but not across actions, nor does it locate *anti-* and *output* dependencies where

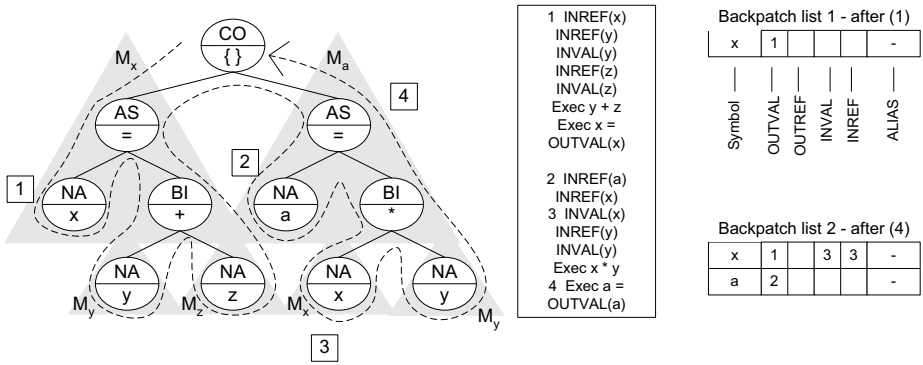


Fig. 2. Semantic execution order, IN/OUT markers and backpatching

symbols are used more than once and must be guarded against race conditions. We extend the IN and OUT sets used by Wolfe [3] to maximise parallelism within each policy action. We also distinguish between value and reference symbols. Consequently, INVAL and INREF refer to retrieved values and references, and OUTVAL and OUTREF refer to changed values and references respectively. OUTREF determines when one symbol *aliases* (or behaves like) another.

Using another depth-first traversal, we simulate sequential policy execution, and generate a list of IN and OUT markings on NA nodes which are used for the analysis of dependencies. Different policy constructs (set/retrieve variable, side-effects, etc) will generate unique lists with combinations of IN and OUT markings. As the list is created, a *backpatch list* is maintained, such that the nodes with the most recent OUT markings and the all IN markings since the last OUT sets are stored. If the symbols are temporary, then IN markings do not need to be maintained, as these symbols are implicitly duplicated. Fig. 2 shows how the IN/OUT markers are generated according to sequential execution order during traversal, and the backpatch lists at point 1 and 4. When OUT markings are encountered, a bidirectional association is formed between the OUT node and the previous OUT node, by inserting a DI data-dependency node. Similarly, a SP flow-dependency node is inserted between IN nodes and the most recent OUT node, to ensure that anti- and output dependencies are sequenced correctly for non-temporary variables. The IN markings are cleared from the backpatch list at each OUT node. Finally, the backpatch list allows symbols to refer to other symbols when aliasing occurs. If policy actions must be sequentially executed, then an SP node is inserted between the partitions on different actions to ensure correct ordering. Fig. 3 shows the dependency tree associations formed for the previous example.

3.3 Parallel execution

Once the policies have been distributed, they can be executed in parallel. Initially, the event source notifies any one of the management nodes involved. This

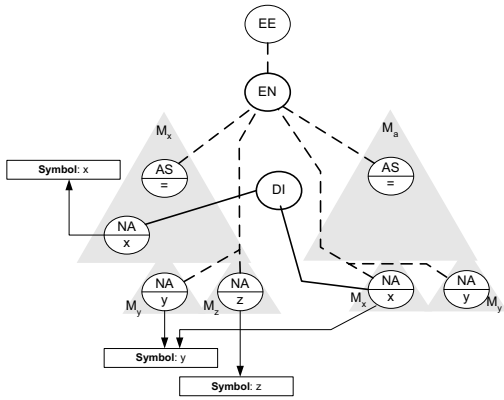


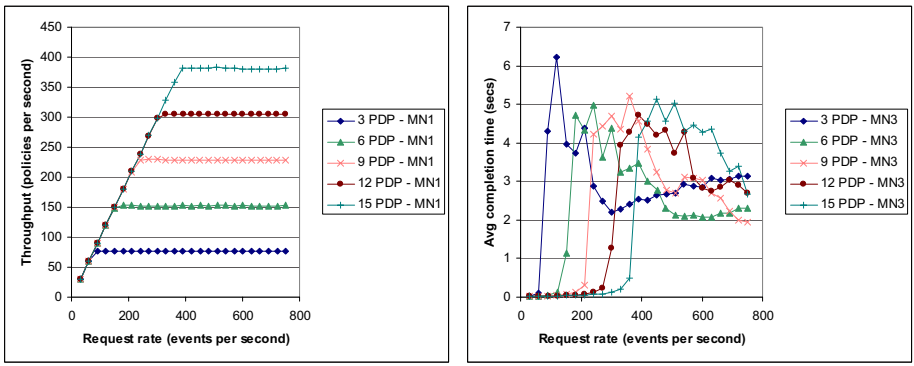
Fig. 3. Dependency tree

management node has primary responsibility for execution, and sends EN messages to the other management nodes to notify them of the event. Any AST partition without SP predecessors can be executed immediately and independently. Partitions with DI predecessors can commence execution, but cannot continue and complete until the relevant data is received from another management node.

Once the partition has completed execution, the management node responsible must then send SP and DI messages to other management nodes when succession is required. These messages indicate that the prior partition has completed execution, and in the case of DI messages, carry the relevant data to be shared between the management nodes.

3.4 Simulation

We have developed a simulation using the OMNET++ Discrete Event Simulator that evaluates the performance and scalability of different types of distributed policies in different network configurations. We model a simple database and multiple management nodes with caching, and describe the different branches of the AST and their relationships using XML. With this simulator, we are able to evaluate this algorithm to determine the impact of different types of policy coordination in large-scale networks. Currently, we use some estimations and some measurements from real systems as timing delays. Early results show the potential scalability of this approach, shown in Fig. 4. Here, increasing the number of management nodes definitely helps in balancing decision load, with only a minimal degradation in response time. Furthermore, the response time under heavy loads is improved. We anticipate that this simulator will be very useful in understanding the cost and performance implications of different architectures for delivering complex personalised services, and help to determine the optimal placement for policy decisions [4].



(a) Throughput (b) Response time

Fig. 4. Scalability

4 Conclusion

We have outlined an algorithm for distributing policies amongst a number of management nodes based on geographical segregation. This allows policies to be distributed outwards to help share the policy decision and enforcement load across many policy-capable servers, to increase the performance and scalability. We create three additional types of nodes for synchronisation and annotate the Abstract Syntax Tree with information that is useful for the execution of those policies in a distributed fashion. Our simulation is able to demonstrate that our approach is helpful in improving the load handling of the policy system as anticipated.

We would like to acknowledge the generous financial support of Alcatel Australia and the Australian Research Council through Industry Linkage Grant LP0219784.

References

1. Sheridan-Smith, N., Leaney, J., O'Neill, T., and Hunter, M.: A Policy-Driven Autonomous System for Evolutive and Adaptive Management of Complex Services and Networks. Presented at Eng. Comp. Based Sys. (ECBS 2005)
2. Grune, D., Bal, H. E., Jacobs, and C. J. H.: Modern Compiler Design. John Wiley & Sons, West Sussex (2000)
3. Wolfe, M.: High Performance Compilers for Parallel Computing. 1st edn. Addison-Wesley, Redwood City CA (1996)
4. Sheridan-Smith, N., O'Neill, T., Leaney, J., and Hunter, M.: Enhancements to Policy Distribution for Control Flow, Looping and Transactions. Internal report UTS-Eng-TR-05-21 (2005)
5. Howard, S., Lutfiyya, H., Katchabaw, M. and Bauer, M.: Supporting Dynamic Policy Change using CORBA Systems Management Facilities. IM (1997)