The Knowledge Engineering Review, Vol. 9:1, 1994, 67–72

Scaling up production systems: Issues, approaches and targets

ANURAG ACHARYA

School of Computer Science Carnegie Mellon University, Pittsburgh PA 15213-3891, USA (email anurag.acharya@cs.cmu.edu)

1 Introduction

Production systems have successfully made the transition from a trendy research idea to a routinely used programming paradigm. An important cause of this transition has been the several orders of magnitude speedup in program execution achieved in the past few years by the combination of better match algorithms, efficient compilation techniques and faster hardware platforms.

A side-effect of the increasing acceptance of production systems is that researchers are investigating their applicability for pattern-directed processing in new application areas. Production systems are being proposed as a single uniform mechanism for diverse tasks in relational database systems—enforcing integrity constraints, monitoring data access and evolution, maintaining derived data, enforcing security schemes, maintaining version histories, implementing alerters and triggers that initiate actions in the presence of specific data patterns (Hanson & Widom, 1993). Integrating production systems and relational database systems is currently a focus of research among database researchers (Hanson, 1992; Widom & Finkelstein, 1989; Stonebraker, 1992; Delcambre & Etheredge, 1988). Many commercial relational databases (e.g. Sybase, Oracle, Rdb and INGRES) provide some, albeit limited, support for production systems. Another new application area is image-interpretation where production systems are used to coordinate and control image segmentation, segmentation analysis and the construction of a scene model. An example is the System for Photo-interpretation of Airports using MAPS (SPAM) system developed at Carnegie Mellon (McKeown et al., 1985). Production systems are also being used in simulation efforts. Among other things, they have been used to simulate a multicache system (Rim & Cragon, 1988).

In addition, traditional application areas like expert systems and modeling human cognition have significantly evolved. Expert systems have moved beyond the knowledge-rich, data-lean domains like configuration and diagnosis to knowledge-lean, data-rich domains like process monitoring (Laffey et al., 1988), analysis of large data sets generated by scientific simulations and imaging as well as knowledge-rich, data-rich domains like logistics and routing. Many contemporary efforts at modeling human cognition aim to develop unified models for human cognition (Laird et al., 1987), unlike the traditional efforts which attempted to model individual cognitive phenomena.

These applications greatly increase the demands on production system implementations. Most of them—active databases, image-interpretation, simulation and data-rich expert systems—process orders of magnitude more data than the traditional applications. Typical databases often have millions of tuples, large databases even more. On the other hand, unified models of human cognition use production systems as models of human memory. To be successful at modeling a truly wide range of phenomena, these models need orders of magnitude more productions. In short, production system implementations need to scale up.

2 Issues

Production systems have been notorious for their inability to handle large amounts of data. Production system programming texts like "Programming Expert Systems with OPS5" (Brownston

et al., 1985) devote several pages to tricks to avoid excruciating slow-downs; programs that deal with large amounts of data have to be modified to allow the data to be processed piecemeal, for example SPAM and Alexsys (Harvey, 1993; Stolfo, 1993). The primary cause of this poor scalability is that the match algorithms used generate large numbers of combinations of individual tuples (Gupta, 1986; Miranker et al., 1987; Tambe & Rosenbloom, 1990). These combinations are generated during the match procedure as intermediate results and as instantiations. The total number of such combinations can be a high order polynomial function of the number of tuples, (Miranker et al., 1990), which leads to a combinatorial explosion as the number of tuples grows. For the sake of efficiency, almost all existing match algorithms that are capable of matching powerful rules maintain some of these combinations as intermediate "match state" (Brant et al., 1991; Gupta et al., 1986). To improve the data-scalability of production systems, it is imperative to tame this combinatorial explosion.

Production systems have not scaled well along the dimension of productions either. Intuitively, this is due to the fact that for every new rule in the program, some tests will have to be performed to determine whether it is applicable in the states the program proceeds through. Even if only a small constant number of additional tests are required per production, there is a linear slow-down as the number of productions is increased. The performance degradation that occurs as the number of productions in a program increases is referred to as the "average growth effect". Significant research effort has been directed at getting rid of it or getting around it (Minton, 1988; Etzioni, 1990; Gratch & De Jong, 1992; Cohen, 1990; Doorenbos et al., 1992). To improve the production-scalability of production systems, it is imperative to limit the growth in the number of tests to a sublinear function of growth in the number of productions.

Production systems are symbolic programs with highly irregular memory access patterns. This becomes increasingly important as processors get progressively faster than memory and main memory becomes further and further away from the processor. Furthermore, as the size of the programs grows (whether due to an increase in data or in productions), this irregularity propagates down the memory hierarchy, eventually to the virtual memory system which leads to thrashing. To improve scalability of production systems, it is necessary to improve their locality properties and to reduce the load on the entire memory subsystem.

3 Taming the combinatorial explosion

Four major approaches have been proposed for taming the combinatorial explosion.

- *Discovering and exploiting implicit structure:* Even though tuple spaces are flat and unstructured, large data sets usually have some structure—some form of grouping is possible. Collection-oriented match is a new approach to matching proposed by Acharya and Tambe (1993) which attempts to discover this structure and take advantage of it as far as possible. The basic idea is to allow individual conditions to match collections of tuples instead of individual tuples, and to split collections if, and only if required by the inter-condition constraints. All known match algorithms that operate on a Rete-like network of conditions can be easily transformed to their collection-oriented analogues. The paper describes a collection-oriented analogue of the well-known Rete matching algorithm. An underoptimized implementation of this algorithm achieved upto several orders of magnitude reduction in execution time and up to one order of magnitude reduction in the amount of space required over an optimized implementation of the Rete algorithm. In one of the programs, the new algorithm matched about 6 million tuples in a few minutes. The gains achieved by this approach depend upon the extent to which tuples can be grouped together. In particular, it does not improve the worst-case performance.
- *Scalable parallelism:* If the available parallelism scales with the amount of data and if there are enough processors in appropriate processor configurations, then parallelism could be used to handle the combinatorial explosion for particular applications. Production system application that process large amounts of data often have a large number of potentially data-parallel operations raising the hope that the available parallelism would scale with the amount of data. Research on parallelizing compilers for existing languages have, however, yielded disappointing results—there appears to be an empirical bound of about 30–40 fold on the amount of

parallelism available in production systems. This limit can be attributed to the sequential nature of the existing languages. Recent research has focused on explicitly parallel production languages (Acharya, 1993a), and has yielded more encouraging results. Results indicate that the parallelism available in many programs indeed scales with the amount of data (Hernandez & Stolfo 1993; Acharya, 1993b). Though these results hold promise, it should be noted that all of them are based on different levels of simulations and not real implementations. Another point to note is that data-parallel operations are carried out on collections of data that match the same production, and are similar in characteristics which implies that there is a synergy between collection-oriented match and scalable parallelism.

- *Limiting expressiveness:* Most production system formalisms are powerful enough to encode NP-Hard problems like subgraph isomorphism for all graphs. Tambe and Rosenbloom (1990) investigated the approach of limiting expressiveness of the production system formalism so that it is possible only to encode polynomial problems in individual productions. In some cases, this yielded dramatic results, in some others the reduction in the expressiveness of individual productions resulted in an exponential blowup in the number of productions needed to achieve equivalent functionality. This offsets the improvements achieved from the limited formalism. Furthermore, it substantially increases programming effort. The collection-oriented match approach was initially conceived of as a way to achieve the gains of this approach without its disadvantages.
- *Limiting match results:* Most existing production languages are sequential. Even though many instantiations could be (and are often) generated every recognize-act cycle, only one is fired. The others may never be fired. Miranker et al. (1990) proposed integration of the instantiation selection policy into the matching algorithm to limit the number of instantiations generated to one—just the instantiation that would be fired. In several of the traditional programs, this approach yielded a significant improvement in execution time. However, the assumption of single instantiation firing goes contrary to the collection-at-a-time data-parallel nature of most of the target applications including databases. However, Acharya and Tambe (1993) note that such algorithms can be transformed to their collection-oriented analogues relatively easily. The transformed algorithms would be able to support collection-oriented semantics without giving up the parsimonious matching of the original approach.

Of the four approaches, collection-oriented match holds the greatest promise, since it has shown very encouraging results without giving up expressiveness and since it can be used on any configuration including uniprocessors. Parallelism would be a great win if appropriate machine configurations would be available. All current work on scalable parallelism assumes a uniform shared memory model which is not scalable. However, this work does establish that for many production system programs, the available parallelism scales with data which encourages investigation of scalable configurations as platforms for production system programs.

Collection-oriented match also holds out promise of additional parallelism as it tries to match collections of wmes against each other. This is an open area for research.

4 Eliminating linear average growth effect

Three major approaches have been proposed to eliminating linear average growth effect. All of them involve transformation of the match network.

- *Sharing:* It is an empirical observation that productions in large systems are not unstructured. Instead, they are organized into clusters and this organization is reflected in common conditions at the beginning of the production that test context information. At any point of time, only a small number of productions usually are applicable and the rest fail to match the context. If all common tests are shared, then adding a new production adds no cost most of the time—since the context test is shared and since most of the time, the rest of the test are not needed. In systems that explicitly maintain goal context, like Soar (Laird et al., 1987) and Prodigy (Carbonell et al., 1990), this can yield large speedups (Doorenbos, 1993). Sharing was one of the initial optimizations provided by the Rete algorithm, but its utility to scalability has only recently been recognized (Doorenbos, 1993). Sharing is a static transformation of the match network and depends only on the structure of the productions.

- *Hashed alpha tests:* As the number of productions in a program goes up, the number of distinct conditions to be tested usually goes up too (note that this is not necessary but is almost always true). This can cause a linear increase in the number of tests needed to determine the set of conditions a particular tuple matches (known as alpha tests). In most cases, this increase can be eliminated by organizing the conditions in a hash table with the values of the constants being tested as arguments to the hash function. To avoid linearization of the hash tables as the number of conditions increase, this hash table has to be dynamically reconfigured as it fills up. The technique was first proposed by Acharya (1992) and has been implemented in the latest version of Soar by Bob Doorenbos of Carnegie Mellon. Experiments indicate that in the set of tasks it has been tested on, it is successful in bounding the number of alpha tests to a small constant. Hashing alpha tests is also a static transformation of the network.
- *Adaptive networks:* A production instantiation is generated if and only if all its conditions are matched. If even one of its conditions is not matched, then any effort spent computing partial matches for the production is wasted. Since only a small number of productions match at any given time, the match effort spent computing partial matches for the other productions (which is most of them) is wasted. The first attempt at trying to adapt the match network to avoid computing partial matches that do not lead to a complete instantiation was the "rule-active" flag introduced by Miranker in the Treat algorithm (Miranker, 1987). The algorithm maintained one such flag for every rule and it was set if all conditions in the rule matched at least one tuple, unset otherwise. However, computation of these flags is linear in the number of affected productions (affected productions being those productions at least one of whose conditions matches a new tuple after a given recognize-act cycle). Since all productions could (and often do) share context conditions, this could take time linear in the number of productions. Another problem with this scheme is that with sharing, computation of the rule-active flag is significantly complicated. Doorenbos (1993) proposes two schemes for locally adaptive networks which avoid the potential linear slow-down associated with the rule-active flag. These schemes unlink and relink nodes from the network depending on the cardinality of the memories that hold intermediate results—if a memory is empty, there are no matches for some section of the production and there is no need to spend any effort matching this production. Unlike, the rule-active flag, this scheme does not conflict with sharing. In successive papers, Doorenbos has demonstrated that these schemes are able to handle hundreds of thousands of production in Soar (Doorenbos et al., 1992; Doorenbos, 1993). All these approaches have been used by Doorenbos in his implementation of Soar version 6. This system has the currently best production-scaling characteristics. However, it assumes that the number of tuples to be matched is small, i.e., it is suitable for a knowledge-rich, data-lean environment. However, none of these schemes conflict with collection-oriented match. Integrating these approaches with collection-oriented match can be expected to work well in all environments.

5 Improving memory subsystem performance

There has been no work specifically towards improving memory subsystem performance of production system programs. To the best of my knowledge, the memory subsystem performance of only one production system program has been investigated and that within the context of an analysis of garbage-collection in lisp programs (Zora, 1989). There has, however, been some speculation about what could be done to improve the memory subsystem performance of production system programs. In addition, there have been several efforts at reducing the memory requirements of production system programs.

- *Reducing the amount of state saved:* Since the set of tuples in a production system usually changes slowly, incremental state-saving algorithms have dominated the scene. Efforts to reduce the amount of state saved attempt to either trade time for space or take advantage of higher rate of change to the tuple space. Points in this space are Treat (Miranker, 1987), which does not save partial matches for groups of conditions, and A'-Treat (Hanson, 1992), which goes one step further by not saving the matches for individual conditions.
- *Reducing the number of instantiations:* Lazy matching (Miranker et al., 1990) attempts to reduce memory requirement by generating only one instantiation at a time.

- *Collection-oriented match:* Takes advantage of the implicit structure in the data to hold the partial matches in a more efficient data structure. This has been shown to reduce the memory requirements in collection-rich cases by upto an order of magnitude.
- *Discardable/recomputable pages:* Recent research in external pagers has indicated that significant gains can be achieved for some memory-starved programs if the operating system interface allows the program to indicate which of its pages do not contain useful data or contain data that can be recomputed (Subramoniam, 1993). This information is used by the virtual memory pager to make better decisions about the best pages to replace when necessary. The intermediate state of production systems is recomputable. While no work has been done to investigate the potential of this virtual memory feature, the size of the recomputable data indicates that this scheme has potential.
- *Better memory allocation:* Like most symbolic programs, production systems have an irregular memory allocation and access patterns. Most allocations are small and tend to be reused in a haphazard fashion. This often leads to a slow down as the execution proceeds (Doorenbos, 1992). For small programs whose working set fits in the cache, this is not a problem. However, as the size of programs increase along either dimension, their working set no longer fits in the cache which forces them to execute at main memory speeds. Furthermore, given the assumption that large data sets have implicit structure (which underlies collection-oriented match and scalable parallelism approaches), I expect memory access would be more predictable for the larger data sets. This is an open area with potential for a lot of research.

6 Targets

Targets have a way of galvanizing research especially if they appear achievable. Given the recent progress in improving scalability of programs, the following targets seem reasonable yet challenging.

- *Data-scalability:* Matching ten million tuples in a reasonable time, where reasonable means minutes and not hours.
- *Production-scalability:* Match one million productions in a reasonable time.
- *Both:* Match a hundred thousand productions against a hundred thousand tuples in a reasonable time.

Achieving these targets will need improvements at all levels—match alogrithms, computational models, implementation techniques, memory subsystem performance, hardware platforms.

References

Acharya, A, 1992. "Hashing alpha tests". Unpublished note.

Acharya, A, 1993a. "PPL: An explicitly parallel language for production systems". In: *Proceedings IJCAI-93 Workshop on Production Systems and their Innovative Applications.*

Acharya, A, 1993b. Recent unpublished results, October.

Acharya, A and Tambe, M, 1993. "Collection-oriented match". In: *Proceedings IJCAI-93 Workshop on Production Systems and their Innovative Applications.*

Brant, D, Grose, T, Lofaso, B and Miranker, D, 1991. "Effects of database size on rule system performance. Five case studies". In: *Proceedings International Conference on Very Large Databases.*

Brownston, L, Farrell, R, Kant, E and Martin, N, 1985. *Programming Expert Systems with OPS5: An introduction to rule-based programming.* Reading, MA: Addison-Wesley.

Carbonell, J, Knoblock, C and Minton, S, 1990. "PRODIGY: An integrated architecture for planning and learning". In: K Vanlehn (ed.), *Architecture for Intelligence* pp 241–278. Hillsdale, NJ: Erlbaum.

Cohen, WW, 1990. "Learning approximate rules of high utility". In: *Proceedings Sixth International Conference on Machine Learning* pp 268–276.

Delcambre, L and Etheredge, JN, 1988. "The relational production language: A production language for relational databases". In: *Proceedings Second International Conference on Expert Database Systems* pp 333–352.

Doorenbos, R, 1992. Personal communication.

Doorenbos, R, Tambe, M and Newell, A, 1992. "Learning 10,000 chunks: What's it like out there?" In: *Proceedings Tenth National Conference on Artificial Intelligence* pp 830–836.

Doorenbos, R, 1993. "Matching 100,000 rules". In: *Proceedings Eleventh National Conference on Artificial Intelligence* pp 290–296.

Etzioni, O, 1990. *A structural theory of search control*. PhD Thesis, School of Computer Science, Carnegie Mellon University.

Gratch, J and DeJong, G, 1992. "A probablistic solution to the utility problem in speed-up learning". In: *Proceedings Tenth National Conference on Artificial Intelligence* pp 235–240.

Gupta, A, 1986. *Parallelism in Production Systems*. PhD Thesis, Computer Science Department, Carnegie Mellon University.

Gupta, A, Forgy, C, Newell, A and Wedig, R, 1986. "Parallel algorithms and architecture for production systems". In: *Proceedings Thirteenth International Symposium on Computer Architecture* June, pp 28–35.

Hanson, E, 1992. "Rule condition testing and action execution in Ariel". In: *Proceedings SIGMOD Conference of Management of Data* pp 49–58.

Hanson, E and Widom, J, 1993. "An overview of production rules in database systems". *The Knowledge Engineering Review* **8** (2) June, 121–144.

Harvey, W, 1993. Personal communication.

Hernandez, M and Stolfo, S, 1993. "Parallel programming of rule-based system in PARULEL". In: *Proceedings IJCAI-93 Workshop on Production Systems and their Innovatice Applications*.

Laird, J, Newell, A and Rosenbloom, P, 1987. "Soar: An architecture for general intelligence". *Artificial Intelligence* **33** (1) 1–64.

Laffey, T, Weitzenkamp, S, Read, J, Kao, S and Schmidt, J, 1988. "Intelligent real-time monitoring". In: *Proceedings National Conference on Artificial Intelligence* pp 72–76.

McKeown, DM, Harvey, WA and McDermott, J, 1985. "Rule based interpretation of aerial images". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **7** (5) 570–585.

Minton, S, 1988. "Learning effective search control knowledge: An explanation-based approach". PhD Thesis, Computer Science Department, Carnegie Mellon University.

Miranker, D, 1987. *TREAT: A new and efficient match algorithm for AI production systems*. San Mateo, CA: Morgan Kaufmann.

Miranker, D, Brant, D, Lofaso, B and Gadbois, D, 1990. "On the performance of lazy matching in production systems". In: *Proceedings Eighth National Conference on Artificial Intelligence* pp 685–692.

Rim, Y and Cragon, H, 1988. "Multicache system simulation using a production system". In: *Proceedings Southeastern Simulation Conference*, October, pp 64–69.

Stolfo, S, 1991. Personal communication.

Stonebraker, M, 1992. "The integration of rule systems with database systems". *IEEE Transactions on Knowledge and Data Engineering* **4** (5), October, 415–423.

Subramaniam, I, 1993. *Managing discardable pages*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.

Tambe, M and Rosenbloom, P, 1990. "A framework for investigating production system formulations with polynomially bounded match". *Proceedings Eighth National Conference on Artificial Intelligence* pp 693–700.

Widom, J and Finkelstein, S, 1989. "A syntax and semantics for set-oriented production rules in relational database systems". *SIGMOD Record*, **18** (3), September, 36–45.

Zorn, B, 1989. *Comparative performance evaluation of garbage collection algorithms*. PhD Thesis, Computer Science Division, University of California Berkeley.