High Performance Declarative Memory Systems through MapReduce

Mark Edmonds, Tanvir Atahary, Tarek Taha
Department of Electrical and Computer Engineering
University of Dayton
Dayton, USA
{edmondsm1, ataharyt1, ttaha1}@udayton.edu

Scott A. Douglass
Air Force Research Laboratory
United States Air Force
Wright-Patterson AFB, USA
scott.douglass.1@us.af.mil

***Abstract*—**This paper describes the acceleration of the declarative knowledge retrieval system of a cognitive architecture, namely ACT-R. The core of ACT-R's retrieval mechanism, activation calculation, is accelerated through leveraging the speed of C++ and the MapReduce program model. Work described in this paper represents an extension of previous Erlang-based concurrent activation. ACT-R's retrieval process is re-examined and optimized in this solution. Concurrency available in the execution platform is exploited to maximize the acceleration of declarative retrieval. The resulting implementation, referred to as Accelerated Declarative Memory (ADM), presents a high-performance activation calculation that enables practical use of more massive declarative memories. ADM presents new mechanisms to access and traverse declarative memory to reduce the overhead of executing retrievals. This solution offers retrieval latencies 20 times faster than the previous Erlang solution.

***Keywords*—**ACT-R; declarative memory; semantic networks; MapReduce; parallel activation calculation.

1. Introduction

Cognitive architectures attempt to computationally describe the functional structure of the human mind. They link the structure of the brain to the function of the mind [1]. The declarative memory module of the ACT-R cognitive architecture gives agents the ability to recall factual information from their past [1]. A growing number of researchers in the cognitive architectures community are investigating ways to increase the capacity and performance of declarative retrieval systems [2]-[6]. Seeking to extend the practicality of existing associative retrieval mechanisms, these researchers are developing algorithms and computational frameworks that support massive stores of declarative knowledge and accelerate knowledge activation calculation.

Capacity and retrieval performance increases emerging from these efforts have the potential to dramatically change the modeling of human memory and the exploitation of declarative knowledge in agent-based software applications. The retrieval system presented in this paper, referred to as Accelerated Declarative Memory (ADM), empowers real-time agents to ask general questions into massive declarative knowledge sources.

ADM expands the practicality of ACT-R's retrieval mechanism through the acceleration of activation calculation. The research effort developing ADM is producing a comprehensive high-performance declarative retrieval system, not just faster activation calculation. The ADM implementation is functionally equivalent to ACT-R's retrieval process and offers substantial performance gains. The paper has the following sections:

Retrieval in ACT-R briefly describes the retrieval process in the ACT-R cognitive architecture. This overview frames the central challenge motivating the development of the ADM retrieval system.

ADM Background describes the origins and motivations of the ADM system. The section explains how ADM and its Erlang-based predecessor (soaDM) represent declarative knowledge in semantic networks and achieve parallel activation calculation through MapReduce.

ADM Technical Details consists of a clear and detailed description of how ADM technically realizes parallel activation calculation through an optimized implementation of MapReduce.

Experimental Setup describes how the performances of soaDM and ADM have been assessed.

Results compares the performance of declarative retrieval in soaDM and ADM.

Conclusion proposes technical and theoretical impacts of the ADM system and suggests future research directions.

2. Retrieval in ACT-R

2.1. ACT-R Overview

ACT-R is a cognitive architecture that can be used to specify and execute computational process models of human cognition [1]. The architecture consists of a central production system and several modules. The central production system can be thought of as the director of cognition. The modules support knowledge processing in the central production system by performing module-specific processes and actions. Modules exist for vision, goal maintenance, situation representation, audition, motor control, and declarative memory [1].

ACT-R's declarative memory is grounded in knowledge representations known as *chunks*. Chunks are composed of key-value pairs that encapsulate a unique piece of explicit factual knowledge. Upon successful retrieval, the chunk with the highest computed activation is placed in a retrieval buffer and can influence the behavior of ACT-R's central production system.

2.2. ACT-R's Activation-Based Retrieval Calculus

The retrieval process in ACT-R is influenced by: (1) *top-down constraints* defined in retrieval requests; and (2) *contextual priming effects* caused by chunks present in

buffers capable of spreading activation. To specify a retrieval request in an ACT-R production, a modeler typically specifies the type of chunk on which to focus the retrieval process and may specify additional top-down constraints that must be met by any successfully retrieved chunk. During the retrieval process, all chunks of the specified type, or derived from the specified type through chunk-type inheritance, are considered initial retrieval candidates. Chunks meeting all top-down constraints defined in retrieval requests are considered final retrieval candidates. The activations of each chunk in the final candidate set are computed, and the chunk with the highest activation is retrieved. The equations governing the retrieval process in ACT-R are listed in Table 1. Chunk *activation* is primarily based on a *base-level* reflecting the prior usefulness of a chunk and *spreading activation* reflecting the degree to which other chunks in context are associated with a chunk. Base-level learning influences retrieval by increasing the activation of recently and/or frequently retrieved chunks. Spreading activation influences retrieval by allowing contextual knowledge to prime chunks through shared knowledge structure and association.

Table 1. ACT-R's activation calculation equations.

Name	Equation
Activation	$A_i = B_i + \sum_j W_j S_{ji} + \varepsilon$
Base-Level Learning	$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right)$
Associative Strength	$S_{ji} = S - \ln(fan_j)$
Probability of Retrieval	$P_i = \frac{1}{1 + e^{-\frac{A_i - \tau}{s}}}$
Latency of Retrieval	$T_i = Fe^{-A_i}$

The activation equation in Table 1 mathematically describes how chunks present in context buffers (indexed by j) produce activation values through strengths of association (S_{ji}) and activation weights (W_j) that are combined with base levels to determine *context-specific activations*. This spreading activation-based retrieval process becomes a computational burden in real-time systems that produce large retrieval candidate sets[1]. To extend the effectiveness of ACT-R's retrieval mathematics to larger declarative memories, the activation calculation must be accelerated.

3. ADM Background

The ADM retrieval system represents a technical extension of the RML1 retrieval system. To take advantage of the large-scale declarative memory system in RML1, cognitive modelers author and execute models in a framework developed using the Erlang programming language [8]. To broaden the usefulness of the RML1 declarative memory system, RML1 was functionally isolated from the broader Erlang execution framework; it was re-implemented as a net-centric software service that can be used in generic service oriented architectures. Again developed using Erlang, this Service Oriented Architecture Declarative Memory (soaDM) provides a declarative memory system that can be controlled and accessed through a published network interface. soaDM is now a critical component of the Cognitively Enhanced Complex Event Processing (CECEP) model specification and execution framework [8].

Several current ARFL research and development efforts are investigating ways to accelerate core components of the CECEP architecture using multi-core and GPGPU architectures [8]. While ADM currently exploits multi-core computers, ultimately it will computationally realize activation-based knowledge retrieval using GPGPUs.

3.1. Declarative Knowledge in Semantic Networks

Both soaDM and ADM represent declarative knowledge as a semantic network. Nodes in the semantic networks represent classes and instances. Edges in the networks represent "object properties" that capture relationships: (1) between classes; and (2) between classes and instances. The semantic network representation of declarative memory enables sub sections of knowledge to be traversed, rather than simply iterating over the entire set of knowledge.

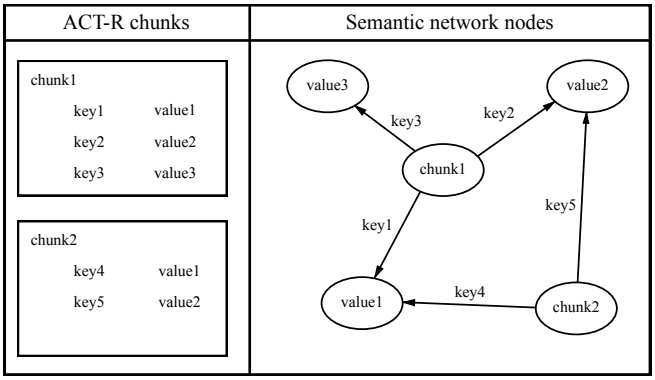

Fig. 1. Comparison of ACT-R chunk representation and soaDM/ADM semantic network representations.

Fig. 1 illustrates the mapping between frame-based chunks in ACT-R and semantic networks in soaDM and ADM. Chunks are realized as nodes, and key/value pairs are realized as directed edges. Nodes internally represent lists of "data properties" that capture relationships between the node and instances of data types (integer, float, etc.). Each node maintains a *fan,* which is a numerical representation of knowledge complexity. In semantic networks, a node's *fan* is the number of nodes with edges that have this node as the head (the number of edges referring to this node). In Fig. 1, *value1* has a fan of 2 while *value3* has a fan of 1.

One of the critical differences between the chunk and semantic network representation is that values used by chunks are expanded into full nodes. They have the same status in the network as the chunk nodes because they are

[1] The number of chunks computing activation correlates directly to the computational stress that the retrieval produces.

literally other chunk nodes. This has powerful implications when attempting to access the network based on a value in a retrieval request.

3.2. Retrieval from Semantic Networks

A retrieval process yielding identical results as ACT-R can be realized in semantic networks using: (1) *activation sources*; (2) *node property filters*. Activation sources specify: (1) *retrieval requirements*; (2) *context primes*.

Retrieval requirements are specified as tuples capturing: (1) a *relation*; (2) a *node* that is in the range of the relation. Node property filters are specified as tuples capturing: (1) a *relation*; (2) either a *node* that is in the range of the relation when it defines an object property or an instance of the relation's data type when it defines a data property. Together, retrieval requirements and node property filters are equivalent to ACT-R top-down constraints.

Context priming sources are specified as tuples capturing: (1) a *relation*; (2) a *node* in the range of the relation; (3) a number corresponding to the *total activation* that that can "spread" from a context source; and (4) a number corresponding to the *structural complexity* of a source context. During retrieval: (1) activation is spread from activation sources; (2) node property filters are applied to nodes receiving activation; (3) nodes that received activation and survive property filters compute their activation; and (4) the node with the highest activation is determined. The retrieval process ultimately returns the set of relations (including the domain and range nodes) originating at the winning node.

Activation calculation in the soaDM mimics the ACT-R's calculation by "spreading" an activation of ***0*** to nodes when a source of activation is a *retrieval requirement* and ***calculated associative values*** when a source is a *context priming source* [2]. ADM's *retrieval requirement* enforcement and *context priming source* activation contributions are functionally equivalent and facilitated through multiple data structures described in Section 4. Fig. 2 shows a part of a semantic network capturing propositions about people and locations.

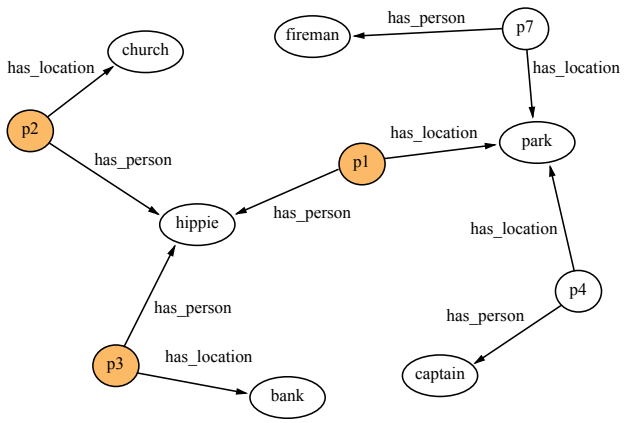

Fig. 2. Semantic network representing propositions {p1, p2, p3} that a *hippie* that *has_location* relations to *church park* and *bank*.

Assuming equivalent chunk knowledge is available in ACT-R, Table 2 shows ACT-R retrieval requests and soaDM/ADM retrieval requirements and node property filters that are functionally equivalent. The first ACT-R retrieval request requires that any retrieved chunk be of type *proposition*. The first soaDM/ADM retrieval requirement spreads activation to all nodes related to *proposition* through the *type* relation. Identical activation calculation across all proposition chunks/nodes occurs in both retrieval systems.

Table 2. Retrieval comparison excluding context priming.

ACT-R retrieval	Retrieval requirements	Node property filters
`+retrieval>` `isa proposition`	`type` `proposition`	

`+retrieval>`
`isa proposition`
`has_person hippie`
	`type`
	`proposition`
		`has_person`
		`hippie`

`+retrieval>`
`isa proposition`
`has_person hippie`
	`has_person`
	`hippie`
		`type`
		`proposition`

The second and third retrieval requests impose an additional restriction on the retrieval process; retrieved chunks must also possess a key/value pair "has_person hippie" (ACT-R) or candidate *proposition* nodes must be related to the *hippie* node via a *has_person* relation (soaDM/ADM). Note that swapping retrieval requirements and node filters in the third comparison spreads activation from the *hippie* node and then requires that any candidate nodes be of type *proposition*. This swap illustrates that: (1) retrieval in soaDM/ADM is not dependent on a type or "ISA" property; and (2) that different activation sources can be used to effect the same retrieval. This latter point is important because spreading activation from a low-fan node such as *hippie* can dramatically alter the complexity of retrieval. If the fan of *proposition* is 1 million, then activation is spread to potentially 1 million candidate nodes when *proposition* is used as a retrieval requirement activation source. Alternatively, if the fan of *hippie* is 25, only 25 candidate nodes are considered to receive spreading activation.

In soaDM, the modeler explicitly requests this swap between retrieval requirements and node property filters. In ADM, the optimal swap is found by querying the network for the node with the lowest fan. ADM then applies the other retrieval requirements as node property filters surrounding the node with the lowest fan. Section 4.2 describes the differences, advantages, and justifications of this process.

Table 3 shows how context priming sources are used in soaDM/ADM to reproduce context-based spreading activation. Assuming the chunk *fireman* is available in one of ACT-R's activation source buffers, the retrieval processes in both systems will yield identical results.

Table 3. Retrieval comparison including context priming.

ACT-R retrieval	Retrieval requirements	Context priming sources	W	N
+retrieval> isa proposition	type proposition	has_person fireman	1	3

The semantic network representation allows for more optimized searching and traversal if activation calculation is effectively managed across candidate nodes. The critical challenge of achieving high-performance with the semantic network approach to retrieval is realizing activation calculation across all candidate nodes as quickly as possible. Both soaDM and ADM use the MapReduce computing model to maximize the concurrency of candidate node activation calculation. The soaDM declarative system utilizes lightweight threads and Erlang's message passing to coordinate the execution of the retrieval process using MapReduce [2]. The ADM declarative system uses hash tables to coordinate and execute an equivalent computing model.

3.3. Spreading Activation Using MapReduce

MapReduce can be described as simplified processing on multiple threads [9]. In Flynn's taxonomy of computer architectures, MapReduce is classified as Single Program Multiple Data (SPMD). In SPMD architectures, multiple independent processors execute the same program on different data. The MapReduce programming model consists of: (1) independent data elements; (2) a computational process that can be applied to each independent data element; (3) processes called "mappers" that apply the computational process to data elements; and (4) processes called "reducers" that integrate computational results reported by "mappers" into a final computational result.

The soaDM and ADM retrieval systems map the following facets of spreading activation calculation to the MapReduce programming model. Both retrieval systems define the aspects of the MapReduce processing pipeline shown in Fig. 3 in the following manner.

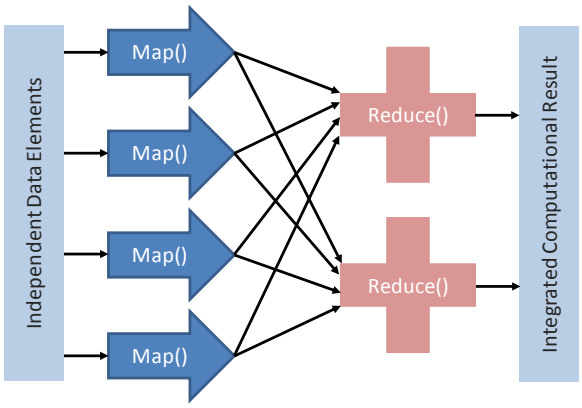

Fig. 3. The MapReduce processing pipeline that transforms data elements into a final result.

Independent Data Elements: processed data consists of semantic network nodes representing: (1) ACT-R retrieval parameters; (2) base level histories; (3) edge collections capturing object properties; and (4) node attributes capturing data properties.

Computational Process: chunk/node activation calculation functionally replicating ACT-R's recency/frequency retrieval calculus. Calculation depends on: (1) base levels; (2) ACT-R declarative module control parameters; (3) top-down constraints defined in retrieval requests; and (4) contextual priming effects.

Mappers: processes that apply activation calculation process over sub-sets of semantic network nodes being processed. Mappers apply activation calculation process to subsets of semantic network nodes corresponding to retrieval candidates and relay activation results back to the main retrieval process through reducers.

Reducers: processes that integrate computational results from mappers into a final result. In soaDM and ADM, a "top-level" retrieval computation receives activation results from the mappers and returns the semantic network node with the greatest activation. This node is the base of the knowledge retrieved from memory.

Both the soaDM and ADM declarative memories consider retrieval candidate sets to be a data set that can be processed while exploiting SPMD parallelism. The use of the SPMD paradigm to achieve concurrent spreading activation calculation distinguishes the discussed work from related work using database technologies [3]-[6]. Both the soaDM and ADM solutions divide the candidate node set into "work units" which are processed through concurrent mappers. Both solutions also "reduce" node/activation computational results in the same general manner.

4. ADM Technical Details

The Accelerated Declarative Memory (ADM) system is based on a MapReduce programming model implemented in optimized C++. This section of the paper describes how semantic networks are implemented using hash tables and how parallel spreading activation and base level calculations are realized in C++ threads using MapReduce.

4.1. Restructuring DM using Hash Tables

ADM utilizes sets of hash tables with references to enable rapid access into any portion of the semantic network. Hash tables were chosen due to their constant lookup time, making network access during retrieval direct and independent of network size. ADM avoids the overhead of iteration or message passing present in the CMU's ACT-R distribution and soaDM, respectively. The desired portion of the network can be directly accessed in a maximum of two hashing functions.

There are two critical hash tables that facilitate network access. The first hash table consists of a Name to Node (NTN) map, which maps node names to a reference with their memory location. The NTN is used to verify the need to add or merge an incoming piece of knowledge and to

spread activation from context priming sources. Adding or merging a node is defined by ACT-R [1], and Section 4.3.1 describes the NTN's role during spreading activation.

The second hash table maps relations to a hash table of nodes with the relation pointing to them. This hash table is a Master Relation List (MRL). The MRL encompasses every relation present in the semantic network. The value stored in each MRL entry is another hash table, namely a Relation List (RL). Fig. 4 shows the structure of the MRL and corresponding RLs, as well as how they access the semantic network.

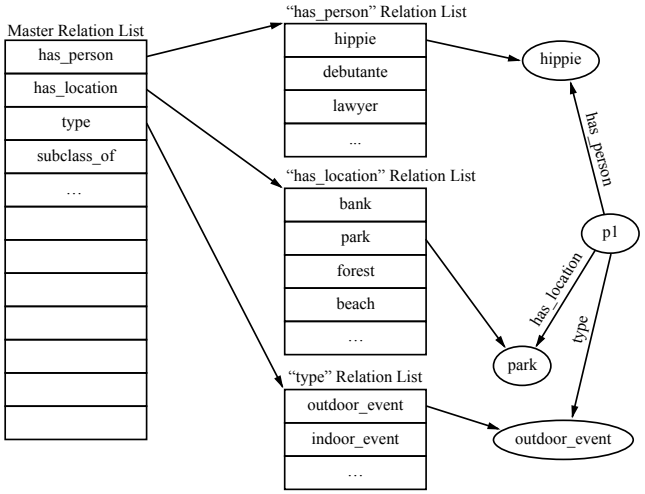

Fig. 4. MRL and RL semantic network access.

Each RL contains the nodes that have nodes referring to them (the head of the edge) through the particular relation. It is critical that the RL be filled with nodes that have the relation referring to them, rather than with nodes referring away from themselves (the tail of the edge) through the relation. The MRL and corresponding RLs facilitate accessing all nodes that contain a relation to a particular node. This represents one of ADM's main performance advantages; ADM can go from a retrieval requirement directly to the relevant portion of the semantic network in constant time.

4.2. Optimizing Candidate Set Determination

Determining the nodes that fulfill all top-down constraints is one of two critical retrieval steps. Candidate set determination showcases one of ADM's primary advantages over ACT-R and soaDM. Note that every node in the candidate set must fulfill every retrieval requirement. The candidate set is *always* the intersection between each set of nodes fulfilling a single retrieval requirement.

Consider the example described Table 2 with "*type proposition*" and "*has_person hippie*" as retrieval requirements. The *proposition* node has a fan of 1 million nodes, and the *hippie* node has a fan of 25 nodes. Note that a node's *fan* is the number of nodes with edges that point to this node (refer to Section 3.1 for details on *fan*). The algorithm described below is executed sequentially.

In Table 2's example, the candidate set is the set of nodes that are connected to *proposition* through a *type* edge and are also connected to *hippie* through a *has_person* edge. Let P_1 = { *x is connected to "proposition"* | *x is connected through "type"* }, H_1 = { *x is connected to "hippie"* | *x is connected through "has_person"* }. Therefore, $P_1 \cap H_1$ represents the nodes fulfilling both requirements and is the candidate set. Suppose $|P_1|$ is 200 nodes, $|H_1|$ is 10 nodes, and $|P_1 \cap H_1|$ is 3 nodes. Fig. 5 depicts these sets.

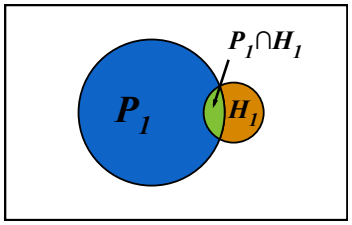

Fig. 5. Intersection of retrieval requirements.

Huge performance advantages could be gained by finding the smallest set of nodes fulfilling a single retrieval requirement (e.g., H_1). Such a solution requires examining node edges and introduces a heavy computational burden to verify the smallest set has actually been found. However, a good approximation can be rapidly verified by finding the node specified in the retrieval requirements with the lowest fan; fan represents the maximum number of nodes a particular node *could* add to the candidate set.

Reexamining Table 2's example, let P_2 = { *x* | *x is connected to "proposition"* }, H_2 = { *x* | *x is connected to "hippie"* }. P_2 and H_2 encompass the total number of nodes connected to *proposition* and *hippie*, respectively. $P_2 \cap H_2$ is the intersection between nodes connected *proposition* and *hippie* through any relation, including *type* and *has_person*. By definition, $P_1 \subseteq P_2$, $H_1 \subseteq H_2$, and $(P_1 \cap H_1) \subseteq (P_2 \cap H_2)$. $|P_2|$ = 1 million nodes (the fan of *proposition*), and $|H_2|$ = 25 nodes (the fan of *hippie*). The exact value of $|P_2 \cap H_2|$ is not important for this example, however $|P_2 \cap H_2|$ must be ≥ 3 (as $|P_1 \cap H_1| = 3$).

Fig. 6 depicts the approximation. Selecting the node with the lowest fan (e.g., *hippie*) reduces the number of nodes that are checked for candidacy. In the example, it is the difference between checking the candidacy of 25 nodes versus 1 million nodes. Regardless of how many nodes are checked for candidacy, only the 3 nodes fulfilling both retrieval requirements ($P_1 \cap H_1$) will be added to the candidate set.

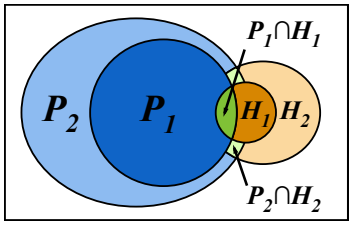

Fig. 6. Intersection of retrieval requirements showing entire fan.

ADM realizes this approximation by iterating over each retrieval requirement, finding the corresponding node through the MRL, and updating a reference to the node with the lowest fan as needed. From the node with the lowest fan (e.g., *hippie*), ADM applies each retrieval requirement as a node filter. All nodes that survive filtering are added to the candidate set. This effectively automates the swap of retrieval requirements and node property filters utilized by modelers using soaDM.

4.3. Retrieval Execution in ADM

To minimize retrieval latencies, ADM supports two retrieval execution modes: serial and parallel. Serial execution is best suited for extremely small declarative memories where the overhead of launching and synchronizing threads outweighs the latency of retrieval. Parallel execution utilizes MapReduce to divide portions of computation across available resources. Parallel execution follows the same algorithm as serial execution but across multiple threads.

4.3.1. Serial Retrieval Execution

ADM's serial operation executes in the following order: (1) nodes are added to the candidate set; (2) activation is spread; (3) activation is computed; and (4) the node with the highest activation is returned to working memory.

1. ADM's serial operation begins by determining the candidate set, as described in Section 4.2.
2. After determining the candidate set, activation is spread from context priming sources. The NTN map facilities finding the nodes in each context priming source, simply by looking them up by name. In Table 3's example, *fireman* is found through the NTN, and nodes connected to *fireman* through *has_person* receive appropriately weighted activation, governed by the equations in Table 1. ACT-R is insensitive to the key in key/value (relation/node) pairs describing context priming sources. ACT-R behavior can be replicated in soaDM and ADM by passing "'*'" as the key [2]. Using ACT-R behavior, every node connected to *fireman* receives appropriately weighted activation.
3. Activation is computed for each node in the candidate set, following the equations in Table 1. The node with the highest activation is returned to working memory.

ADM's serial execution describes how semantic networks can realize ACT-R's retrieval process utilizing hash tables for network access. The serial execution algorithm is utilized during parallel execution, but with the MapReduce computing model to divide computation.

4.3.2. Parallel Retrieval Execution

Candidate set determination and activation computation can greatly benefit from MapReduce. Each node in the network operates as an independent data element. The ADM main thread launches two threads to begin the concurrent retrieval process: (1) a candidate set manager fill the candidate set based on retrieval requirements and (2) a spreading activation manager to spread activation from context priming sources. In ADM's MapReduce model, manager threads act as reducers, and worker threads act as mappers. Fig. 7 shows ADM's parallel retrieval execution. Each "Sync" line in Fig. 7 represents thread synchronization due to data dependence before advancing execution. The algorithm for ADM's parallel execution is below in Fig. 7.

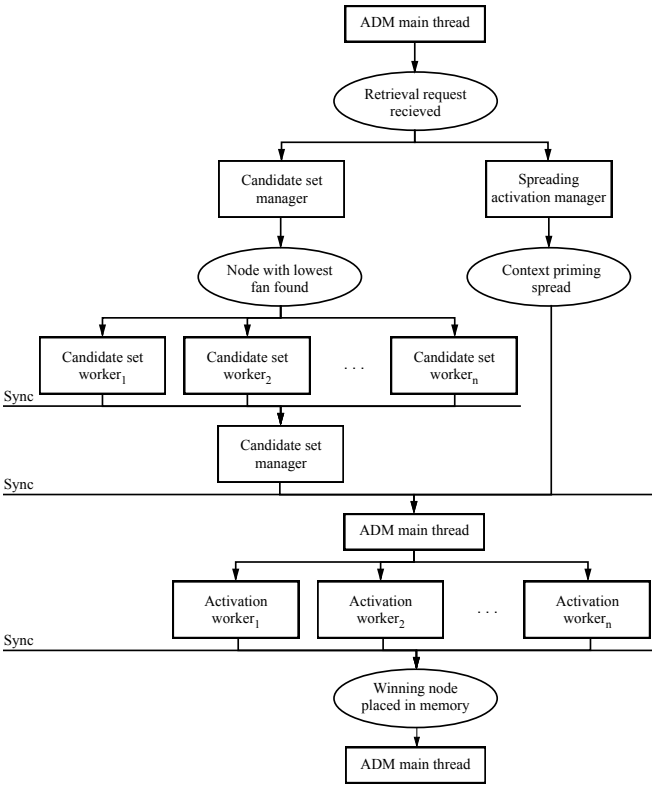

Fig. 7. ADM parallel retrieval execution.

1. The candidate set manager processes retrieval requirements to fill the candidate set and acts as the reducer for the candidate set determination. The node with the lowest fan is found by iterating over the retrieval request as described in Section 4.2. The candidate set manager launches worker threads based on the structure of the network surrounding the node with the lowest fan. The number of workers launched is the number of processor cores available in the system. Each worker is assigned a portion of the relations coming into the node with the lowest fan. Upon completion of each worker, the candidate set manager collects the candidates reported by each thread into the final candidate set.
2. The spreading activation manager runs concurrently with the candidate set manager and distributes activation from context priming sources. The execution of spreading activation follows the description in Section 4.3.1. Only one thread is reserved to spread activation due its simple computation.
3. Once the candidate set and spreading activation managers finish, execution returns to the main ADM thread. Every node in the candidate set must re-compute

its activation. Activation calculation is split among worker threads to saturate all available processor cores. Each worker launched is responsible for computing a section of the candidate set. Each thread reports their winning node, and the ADM main thread returns the node with the highest activation to working memory.

During candidate set determination and spreading activation, more threads than available processor cores are in execution. The execution of candidate determination and spreading activation is highly variable based on the structure of the network and the specific retrieval request. However, determining candidates is a substantially heavier computational burden than spreading activation. To maximize CPU saturation during the entire retrieval process, the number of threads launched overshoots the number of processor cores available by 1. The additional thread is reserved for spreading activation. ADM relies on the operating system scheduler to appropriately allocate resources.

5. Experimental Setup

In order to meaningfully compare soaDM and ADM, the analysis described by [2] was replicated. The testing computer had an Intel Xeon W5590 with 48GB (12x4GB) of 1333MHz RAM running Ubuntu 14.04 LTS. The declarative memory parameters and retrieval requests were identical in all tests. The ontology sources, number of nodes computing activation, and retrieval requests are shown in Table A1. Because each retrieval request contained only one retrieval requirement, the speedup ADM produced is due to ADM's optimized MapReduce model, not from the algorithm to select the optimal starting node outlined in Section 4.2.

6. Results

Fig. 8 concisely shows that ADM achieved a roughly 20 times performance increase over soaDM.

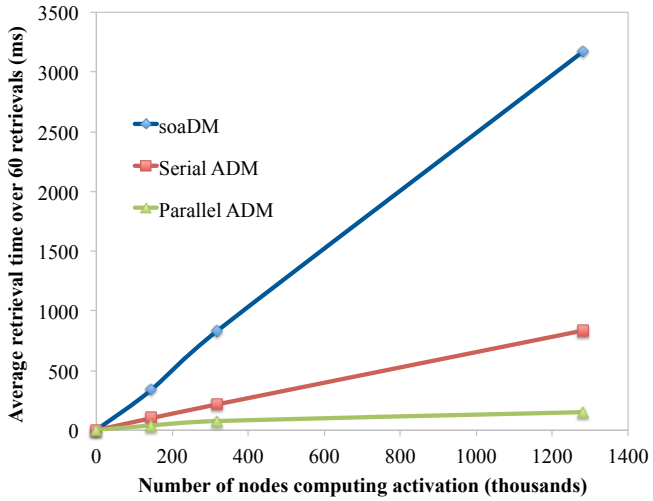

Fig. 8. Performance comparison of soaDM and ADM.

ADM was able to coordinate the full ACT-R activation calculation across 1.3 million nodes in less than 200ms. Parallel ADM performed approximately 6 times faster than serial ADM over 1.3 million nodes computing activation. ADM's serial execution was optimal when 15 nodes computed activation.

Each retrieval was executed 60 times, and Fig. 8 shows the average over the 60 retrievals. It is worth noting that soaDM is inherently parallel; Erlang automatically distributes activation calculation among available resources. The evaluation between soaDM and parallel ADM is the appropriate performance comparison.

6.1. ADM Comparison to other DM Systems

ADM utilizes a highly optimized MapReduce programming model. The data structures, parallel processing, and execution were specifically designed to support the ACT-R retrieval calculus. ADM performs the full ACT-R retrieval process. SemMemDB from [6] illustrates an interesting method to compute spreading activation in database systems. However, SemMemDB offers no apparent method of enforcing top-down constraints. soaDM enforces top-down constraints by spreading an activation of 0 *only* to nodes connected along the relation specified in the top-down constraint, as described in [2]. SemMemDB appears to only be capable of spreading activation, including 0, to every node connected to the context-priming source node, rather than only the nodes connected through the specified relation. ADM realizes the same top-down constraint enforcement through the MRL.

To understand the performance advantages ADM presents over soaDM, the Erlang-based soaDM needs to be revisited. soaDM facilitates network access and activity through supervisor threads [2]. These supervisor threads utilize Erlang's built-in message passing to wake the desired portion of the semantic network. However, Erlang's message passing requires each message being copied from sender to receiver. Locks, mutexes, and shared memory are not required in this model at the expense of copying data with every message passed. The exact messages sent during soaDM's retrieval are described in [2].

The MRL and NTN enable direct access to subsections of the network at a time, similar to soaDM waking subsections of the network. However, ADM avoids the overhead of message passing; each incoming retrieval request is treated as static, read-only data. ADM also benefits from the speed C++ provides through direct memory references. The MRL and NTN traverse the network using references; accessing network routes is incredibly inexpensive. Shared, read-only data is routed throughout ADM's execution using references. Each directed edge in the network is achieved through a reference to the head node and tail node. These design decisions enable ADM's rapid network traversal.

The algorithm in Section 4.2 shows ADM will automatically find the optimal starting node (the node with the lowest fan) for determining the candidate set. From this perspective, the distinction between retrieval requirements

and node property filters is diminished in ADM. The modeler no longer needs to be concerned with the structure of the knowledge to minimize retrieval latency. It is critical to note that ADM finding the optimal starting node only presents an advantage over soaDM when: (1) there is more than one retrieval requirement and (2) the modeler does not utilize or does not optimally swap retrieval requirements and node property filters.

7. Conclusion

ADM constitutes another step towards creating a declarative memory that can store and retrieve knowledge on massive scales while maintaining real-time performance. The use of a semantic network and MapReduce offers substantial performance gains over traditional ACT-R. ADM optimizes the retrieval process in multiple ways: (1) enabling constant time lookup of any section of the network; (2) traversing knowledge through references rather than iteration or message passing; and (3) guaranteeing the optimal starting node for candidate set determination.

ADM constitutes a complete reconsideration of how to store and access declarative knowledge to lower retrieval latencies. Access to larger declarative memories gives agents the ability to react appropriately in real-time to a wider variety of situations. Cognitive modelers will be able to build applied models and agents that employ declarative knowledge bases composed from sources such as OpenCyc, WordNet, domain ontologies such as the Suggested Upper Merged Ontology (SUMO), and the semantic web.

Future work will focus on parallelizing candidate set determination and activation calculation in specialized hardware. ADM is currently being implemented to harness a GPGPU through CUDA. Leveraging the parallelization present in GPGPUs will further increase the scale of knowledge that ADM can support. ADM's serial, parallel, and CUDA implementations will be combined to minimize the retrieval time. Future work should also include a comparison of the optimized starting node for candidate set determination described in Section 4.2. The implementations presented in [2]-[6] should be carefully evaluated and compared against ADM's retrieval optimizations.

Acknowledgements

Described research was partially supported by the Air Force Office of Sponsored Research (AFOSR) Repperger internship program and the Department of Energy Oak Ridge Institute for Science & Education (ORISE) program.

References

[1] Anderson, J. R. *How Can the Human Mind Occur in the Physical Universe?* New York: Oxford University Press. 2007

[2] Douglass, S. A. & Myers, C. W. "Concurrent knowledge activation calculation in large declarative memories." In D. D. Salvucci & G. Gunzelmann (Eds.), *Proceedings of the 10th International Conference on Cognitive Modeling* (pp. 55-60). Philadelphia, PA. 2010

[3] Douglass, S., Ball, J., & Rodgers, S. "Large declarative memories in ACT-R." In *Proceedings of the 9th International Conference of Cognitive Modeling* (paper 234). 2009.

[4] Derbinsky, N., Laird, J. E., & Smith, B. "Towards Efficiently Supporting Large Symbolic Declarative Memories." *Proceedings of the 10th International Conference on Cognitive Modeling*. Philadelphia, PA. 2010.

[5] Salvucci, D. D. "Endowing a cognitive architecture with world knowledge." *Proceedings of the 36th Annual Meeting of the Cognitive Science Society*. Quebec City, Canada. 2014.

[6] Chen, Yang, Milenko Petrovic, and Micah H. Clark. "SemMemDB: In-Database Knowledge Activation." *Proceedings of the 27th International Florida Artificial Intelligence Research Society Conference*. 2014.

[7] Logan, M., Merritt, E. & Carlsson, R. *Erlang and OTP in Action.* Manning Publishing Co. Stamford, CT. 2010.

[8] Taha, T. M., Atahary, T. & Douglass, S. A. "Hardware Accelerated Cognitively Enhanced Complex Event Processing Architecture," *Proceedings of the 14th IEEE/ACIS,* Honolulu. 2013.

[9] Dean, J., & Ghemawat, S. "MapReduce: simplified data processing on large clusters." *Communications of the ACM*, 51(1), 107-113. 2008.

Appendix

Table A1. Retrievals executed during soaDM and ADM comparison. Each retrieval was executed 60 times. The Fan Effect ontology is provided by the CMU ACT-R distribution. MobyII Thesaurus sources were segmented as described in [2].

Ontology source	Nodes computing activation	Retrieval requirements	Context priming sources	W	N
Fan Effect	15	type event	has_person hippie has_location park	1	3
Moby II (1 in 3)	145,073	type synonym_relation	'*' taxing '*' demanding '*' straining	1	4
Moby II (1 in 2)	318,435	type synonym_relation	'*' mazy '*' whimsical '*' flighty	1	4
Moby II (1 in 1)	1,281,763	type synonym_relation	'*' mazy '*' whimsical '*' flighty	1	4