Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 419 – 428

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

An Extensible Digital Library Service to Support Network Science

S.M.Shamimul Hasan[a,b], Keith Bisset[b], Edward A. Fox[a], Kevin Hall[b], Jonathan P. Leidig[c], Madhav V. Marathe[a,b]

[a]*Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, USA*
[b]*Network Dynamics and Simulation Science Laboratory, VBI, Virginia Tech, Blacksburg, VA 24061, USA*
[c]*School of Computing & Information Systems, Grand Valley State University, Allendale, MI 49401, USA*

Abstract

Network science research aims to understand the underlying properties of complex networks. Large-scale modeling and simulation is the core of network science research. Existing systems take a long time to run large network science experiments with high performance computing resources. Scientific data management systems currently lack the performance efficiency needed to support this type of computation and data-intensive research. Memoization provides the ability to index, archive, and reuse frequently requested and expensive to re-compute datasets. In this paper, a domain independent memoization service to increase the computational execution process performance within cyberinfrastructure-based systems is described. The service is formulated as an extensible memoization framework for the computational and simulation network science domains that is built on top of well-defined metadata objects. We present extensible concepts, discuss the proposed algorithm and framework architecture, and examine the flexible nature of the framework. The framework was utilized as a part of the CINET cyberinfrastructure-based digital library (DL). Our experimental results indicate an increase in the efficiency of the system and recommendation of the service's inclusion in scientific DLs.

Keywords: Algorithm; Cyberinfrastructure; Memoization; Network Science

1. Introduction

Scientific domains are experiencing unprecedented growth of vast interlinked systems called "complex networks." These require rigorous analytics to solve many real world problems. Network science research aims to understand the underlying principles of complex networks. This branch of science has received much attention in recent years due to its interdisciplinary problem solving characteristics. Today, researchers conduct a variety of simulation-based experimentation on complex networks. Large-scale simulation is the core of network science research. This research requires computing resources including high performance computing (HPC), large data repositories, and domain-specific user interfaces. However, different researchers conduct similar simulation studies. Design and completion of simulation-based experimentation requires tedious human effort, which may take days or weeks to execute on large HPC systems. Storing and reusing studies eliminates unnecessary calculations, which speeds-up the execution process and throughput. This directly improves the efficiency of utilized national

*Corresponding author, Marathe. Tel.: +1-540-231-8252 ; fax: +1-540-231-2891.
E-mail address: shasan2@vt.edu (S.M.Shamimul Hasan), kbisset@vbi.vt.edu (Keith Bisset), fox@vt.edu (Edward A. Fox), khall@vbi.vt.edu (Kevin Hall), jonathan.leidig@gvsu.edu (Jonathan P. Leidig), mmarathe@vbi.vt.edu (Madhav V. Marathe).

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.205

and institutional supercomputing resources [1]. The process is similar to the dynamic programming technique of memoization. A research domain may use memoization to store input configurations and computational results for given software application versions. If input configurations of new experiments match previously computed study designs for a specific software application, then stored results are used to prevent recomputing costs [2].

Motivation: Programmers from various fields have not historically used domain-specific memoization services to speed-up simulation execution processes. This type of memoization service has dependencies between the simulation application versions and underlying datasets that arise through the simulation process. The information storage and retrieval community gives little attention to large-scale computation-based experimentation systems. Domain-specific implementations will not manage content generated from multiple applications or cyberinfrastructure (CI) based systems. Domain-independent memoization requests require an extensible memoization service.

Contributions: We propose an extensible memoization framework for network science. The major contributions of our work are as follows.

- **Extensible Framework:** An extensible memoization service required to support network science was conceptualized. Clear description of the proposed framework are provided along with preprocessing, matching, and storing steps. Deployed systems based on this framework support the ability to plug in multiple scientific domains and simulation systems, without the need to modify deployed DL systems.
- **Advanced Capabilities:** Our framework and implementation performs the following five major operations: 1) index and archive metadata regarding simulation-based experiments, 2) retrieve the results of an experiment, 3) retrieve metadata regarding an archived experiment, 4) update experiments and metadata, and 5) remove archived experiments. These functions combine to yield a DL memoization service.
- **DL Service:** Scientific simulation-based DLs require new services. Memoization services do not exist in current DL software. Despite large parameter spaces, common simulations are often requested and recomputed by different researchers. Datasets with large storage footprints can be regenerated so as to reduce long term storage costs. Memoization provides the ability to index, archive, preserve, curate, and reuse content. The proposed framework was implemented as a service in a cyberinfrastructure-supporting DL. A detailed study on the effectiveness and efficiency of the proposed framework and implementations was conducted.

To the best of our knowledge this is the first attempt to design and develop an extensible memoization service for network scientific research. The rest of the paper is organized as follows: Section 2 presents the literature review, Section 3 consists of the CINET overview, our extensible memoization framework is presented in Section 4, Section 5 discusses the implementation, Section 6 reports experiments and results, and Section 7 concludes the paper.

2. Literature Review

HPC systems rarely utilize simulation reuse mechanisms for the systemic support of large-scale scientific efforts. Some researchers actively use temporary memoization to implement dynamic programming within applications. Researchers propose caching intermediate results mechanisms that derive incremental programs from non-incremental programs [3]. [4] proposes a formal model of function caching and practical cache replacement strategy. In [5], authors discuss a functional programming language with function call dependencies and caching of expensive function calls. Others propose a staged monadic combinator library for memoization functions [6]. [7] describes an automatic memoization scheme for software engineering. Costa et al. propose a dynamic trace memoization technique that uses memoization tables to skip the execution of redundant instructions [8].

A number of researchers investigate partial memoization techniques. [9] introduces partial memoization of concurrency and communication. [10] mentions several implementations of memoization for partial evaluation. [11] shows that memoization can help improve performance and power consumption in multimedia systems. Also, [12] presents memoization techniques for reducing power consumption of caches. Ikegaya et al. proposes a memoization technique that depends on reuse and margin speculative multithreading based on value prediction into parallel computations [13]. Acar et al. presents a selective memoization framework to empower programmers

with equality, space, and dependencies in support of application development [14]. [15] details "how constraints can be propagated in a memoizing parser (such as a chart parser) in much the same way that variable bindings are, providing a general treatment of constraint coroutining in memoization." [16] uses memoization in self-triggered control. The FastSim simulator uses memoization to speed-up performance [17]. Borodin and Juurlink apply memoization for fault detection [18]. A number of researchers use memoization techniques in database research [19, 20].

Scientists generally produce large quantities of data, however, the DL community successfully supports a limited number of scientific research efforts. Examples of scientific data management projects include earthquake simulation repositories [21], embedded sensor network DLs [22], and D4Science II [23]. Current simulation-based research is computationally, data, temporally, and human effort-intensive. There is a great need for an efficient network science DL that builds upon an integration of scientific content, communities, and services.

3. CINET Overview

This paper focuses on an extensible memoization framework to support network science and simulation based research. The DL component of the CyberInfrastructure for Network Science (CINET) project [24] implements the framework. CINET includes a web portal providing a computational and analytic environment for the network science researcher, educator and student. The CI part of CINET is responsible for coordinating the interactions between the DL, user interfaces, resource managers, data brokers, and execution broker components. Within CINET, a blackboard is used as a central communication mechanism that provides asynchronous, loose coupling of system components. Different types of requests such as execution requests, DL requests, and data requests can be placed in the blackboard. The resource manager determines the appropriate resource for a request, monitors the health and load of compute resources, and provides a sandbox environment in the shape of a virtual machine for requesting components and untrusted software. Brokers frequently communicate with the blackboard and use corresponding components to fulfill service requests. CINET uses HPC resources to service experiment execution requests. CINET hosts the Granite system [25] and graph dynamical systems calculator (GDSC) system [26] as public use applications. CINET provides many realistic graphs for analysis. Galib, NetworkX, and SNAP are the computational engines that provide the capability to analyze different properties of the graphs. See [24] for more information regarding CINET.

4. Proposed Extensible Memoization Service Framework

4.1. Concepts

The extensible memoization framework is based on formally defined key concepts. A domain is a collection of software components that implement specifics (e.g., Granite and GDSC). Domains are defined as $D = \{d_1, d_2, ..., d_n\}$, where $d_i \in Coll$. Here, $Coll$ means collection of software components. Text labels are used to represent domain information. To store a network science experiment, standard experiment characteristics called attributes are recorded. "Size=10MB" is an example of an attribute where "Size" is the key and "10MB" is the value. Sample simulation attributes are systemIdentifier, date, mimeType, description, platform, person, and size. Attributes must be atomic, strongly typed, and uniquely identifiable. Experiment terms, collection related terms, and Dublin Core terms [27] define the attribute list. Dublin Core, a metadata standard, stores digital objects in a structured manner. Let $AK = \{ak_1, ak_2, ..., ak_n\}$ be a set of attribute keys, and each key ak_i has a valid set of values AV_i. Attribute $ATT = \{(ak_i, av_i)^+ : ak_i \in AK, av_i \in AV_i\}$. Text strings are used to store attribute information. The attribute list is defined based on experiment terms, collection related terms, and Dublin Core terms [27]. A memo is the metadata information of an experiment. The DL stores each experiment as a memo. It has unique ID ($ID \subseteq \mathbb{N}$), number of attributes, and domain information. A memo is considered a basic unit of the DL. A memo can be defined as a 3-tuple $MEO = (id, ATT, d_i)$. A query is an experiment request that provides a mapping of key-value pairs. An example of a query is a network science experiment request to execute "Graph=Miami Network" with a given algorithm. Here "Graph" is the query key and "Miami Network" is the value. Let $QK = \{qk_1, qk_2, ..., qk_n\}$ be the set of keys. Each key qk_i has a set of valid values QV_i. A query QER is defined as $\{(qk_i, qv_i)^+ : qk_i \in QK, qv_i \in QV_i\}$. Requested key and value pairs map to the archived system defined

key and value pairs. Attributes and queries have some similarities. However, queries deal with characteristics of the experiment request and attributes deal with the characteristics of the experiment itself. For memoization, experiment metadata and results must be stored after completed software executions. This process is considered an Event. An event data structure contains domain information that triggered the event and the list of attribute information. Event EVN is a tuple, defined as $EVN = (d_i, EATT)$, where $EATT = \{(x, y) : x \in QER, y \in ATT\}$. Resource Definition Framework (RDF) triples are utilized for storing data. It is a schema-less data storing technique where data are stored in a triple format. Each triple contains subject, predicate and object information. As an example, the fact "Digital object 3's size is 10MB" is represented as triple (Subject: 3, Predicate: Size, Object: 10MB). Attribute keys are used as a vocabulary for the predicates of RDF triples. The RDF triplestore $RDFDB$ is a database that contains triples $TRI = (Subject, Predicate, Object)$.

4.2. Research Questions

Several open questions spur development of the extensible memoization framework. What scientific data does a framework need to handle? How can data be stored? What sort of search queries can the users ask of this data (or the system), and what language will be used to ask the query? What algorithms are used for matching? What is the meta-algorithm for memoization service? The questions are addressed in the following framework section.

4.3. Framework

The major contribution of the paper is an extensible memoization service framework. The memoization layer of the framework is built on top of a metadata layer. It provides a communication platform to manage memoization requests and retrieve results. Different types of domains can submit execution requests. For new domains, only the plugin part of the framework needs to extend the client, query parser, and event handler modules. This powerful mechanism makes our framework capable of supporting multiple domains. The framework can be logically partitioned into different parts and a set domain logic can be added or removed without affecting the core pieces of the system. Figure 1 presents a high level overview of the framework. Preprocessing, matching, and storing are three major stages of the framework. The following is a detailed discussion of each stage.

4.3.1. Preprocessing:

The first stage of data pre-processing ensures that data is in the correct form for framework consumption. Possible dataset types are identified in the initial stage. Commonly available network science tools work with graphs and analysis algorithm modules called measures to perform experimental studies. The content to be managed includes network and measure data, complex experimental input configurations, corresponding results, analyses, statistical plots, and related documents generated from the experiment. For each type of content, the framework should handle appropriate metadata. As an example, network graphs require descriptive metadata such as graph name, description, number of nodes, number of edges, file path, network type, graph format, resource identifier, creator, category, date, contributor, relation, and sources. Software required metadata partially including measure name, description, parameters, command, category, and version. Functions handle software package metadata, identify duplicate or overlapping simulation scenarios, track simulation input and environment variables, and provide performance data. Experimental input and result files must be stored along with their associated metadata.

The client side of a given tool submits an experiment request. The associated digital object is a query. Next, the query parser is responsible for parsing and handling a query. It takes domain and query information and parses the query based on key matching. Our framework is capable of containing multiple query parsers.

4.3.2. Matching:

The metadata index uses the RDF triple information from the earlier steps generated for indexing and querying. An incoming study requests first queries the content collections for existing results. If a match is found when querying, the user will receive the results without further computation. Otherwise, the CI will execute the appropriate software and archive the result for future reuse. Our framework returns all the available execution records. A high level overview of our process is described in Algorithms 1, 2, 3, and 4.

Algorithm 2 takes event information as the input while an event handler processes each event. The event handler takes event information and checks which particular domain generates the event. The handler then prepares

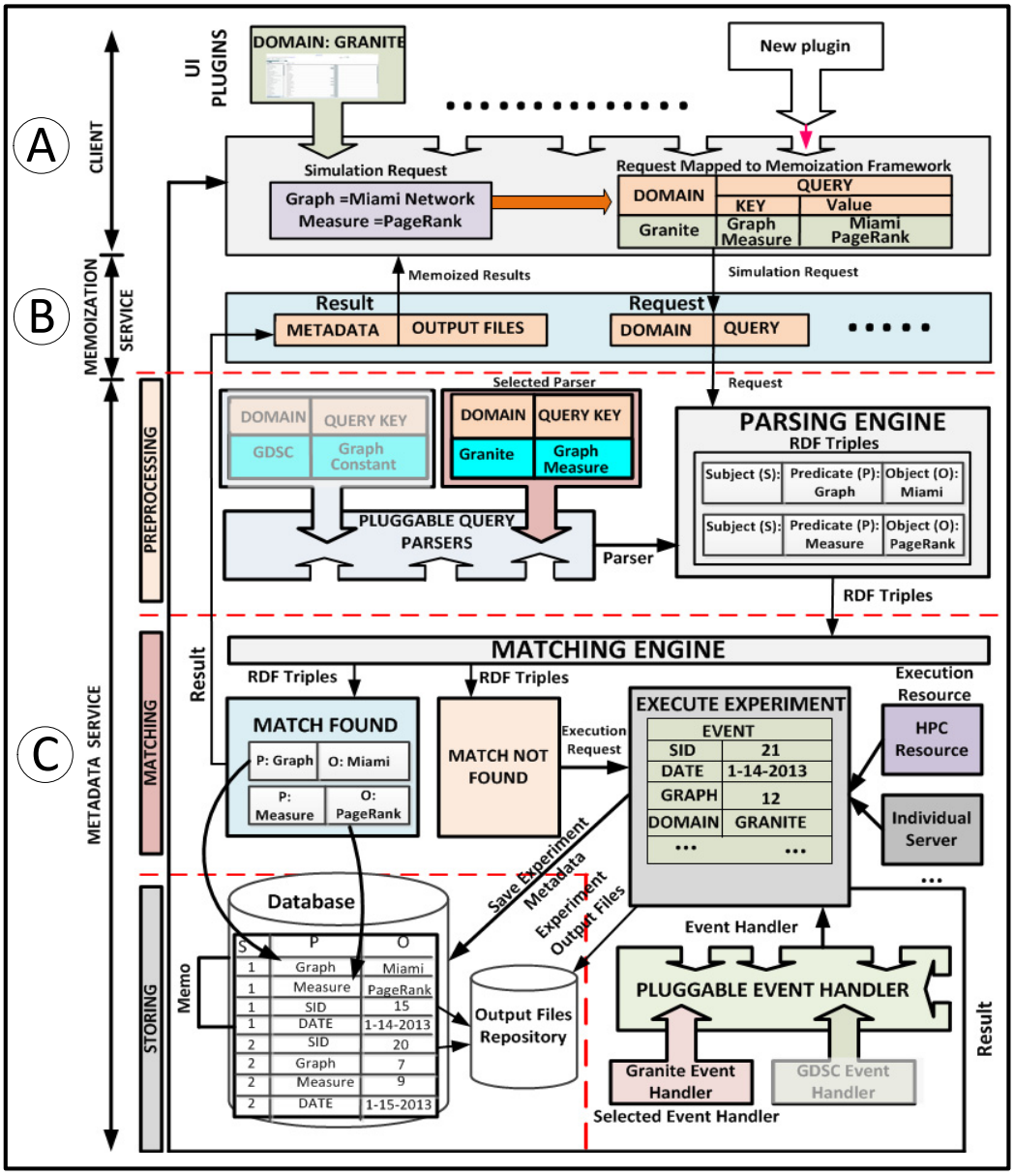

Fig. 1. High level overview of the memoization framework.

metadata with domain, query and attribute keys and values. It prepares RDF triples, saves the metadata, and adds output files to the database.

A system administrator or privileged user can utilize the REMOVE-METADATA algorithm (Algorithm 3), within a curation service, when wanting to remove an archived experiment execution or metadata from the system. The algorithm removes a specific memo from the RDF database. Users can also update the experimental metadata and content for memos, see Algorithm 4. Reverse Polish notation (RPN) technique [28] is used within the DL service when performing all of the queries above.

Algorithm 1 MEMOIZATION: Pseudo-code for memoization service.

INPUT: domain $d \in D$ and query $qer \in QER$.
OUTPUT: List of memo MEO.

send d and qer to the metadata service
$par \leftarrow$ select the query parser $QERPAR$ that can handle the query qer
$(key, value) \leftarrow$ parse the query qer by using parser par
generate $RDFTriple$ using $(key, value)$
compute memo ID $MEOID$ from RDF database $RDFDB$ using RDF
if $MEOID \neq null$ **then**
    provide list of MEO and new execution option to the user.
    if user selects new execution **then**
        $EVENT - REGISTRATION(EVN)$
    else
        $ms \leftarrow$ user selects one specific memo MEO from the provided list
        compute MEO by using ms on RDF database $RDFDB$
    end if
else
    $EVENT - REGISTRATION(EVN)$
end if

Algorithm 2 EVENT-REGISTRATION(EVN): Pseudo-code for registering event.

INPUT: new event: $evn \in EVN$

event handler $EVNHAN$ register the event by performing the following steps.
if event domain is not available in the $RDFDB$ **then**
    save domain on $RDFDB$
end if
compute MEO with $d \in EVN$ and $ATT \in EVN$
save memo MEO to RDF database $RDFDB$

Algorithm 3 REMOVE-MEMO (MEOID): Pseudo-code for removing a memo.

INPUT: Memo ID $MEOID$
OUTPUT: Memo MEO information that removed from RDF database $RDFDB$.

compute memo MEO by using memo ID $MEOID$ on RDF database $RDFDB$
remove memo MEO from RDF database $RDFDB$

Algorithm 4 UPDATE-MEMO: Pseudo-code for updating a memo.

INPUT: memo MEO

compute memo ID $MEOID$ from memo MEO
send memo ID $MEOID$ to $REMOVE(MEOID)$
save memo MEO to RDF database $RDFDB$

In the network science domain, researchers actively work with approximation algorithms to identify approximate solutions to optimization problems. Multiple results from approximation algorithms may be available in the database. Suppose for a particular approximation algorithm, a user is interested in solutions within 10% of the optimal. The framework will take that constant factor information and will return the match where the optimal solution constant is <10%. To support the above filtering option, optimal solution constants need to be added as a query key.

4.3.3. Storing

A central repository stores and archives the memoized experiments. Structured, semi-structured, or unstructured database techniques can be employed to store the experiment information along with simulation output files. An RDF triplestore technique is ideal for this implementation due to its flexibility with dynamic metadata schemes in comparison to relational databases. The schemaless nature allows research groups to map any simulation application to the RDF triplestore without customizing relational DB tables. RDF triplestore allows administrators to modify metadata schemas for individual contexts as well as index or search content without changes to a structured data model or DL.

5. Implementation

The framework was implemented within the CINET DL services. CINET DL is developed using the 5S formal framework [29]. The 5S framework provides a powerful mechanism to manage a complex information system. First, a client submits an execution request through the UI, and the blackboard creates a corresponding entry. The resource manager processes the request to check for any existing, identical study. The resource manager creates a DLRequest in blackboard, which is picked up by the DLBroker. The request is sent to the appropriate DLService to fulfill the request. In this case, the DLService selects the memoization service. Based on memoization arguments, the memoization service queries the database and provides a list of retrievable results. If the experiment does not already exist, the resource manager creates an execution request. The execution broker recieves the request and performs the novel execution. After execution, the memoization index saves the results and sends them to the user. Even if the user requested experiment is available in the memoization DB, the user may optionally perform a fresh execution.

The implementation is divided into four parts; core, plugins, metadata service, and memoization service. Memoization uses the metadata service to serve requests. Different domains can submit execution requests. Domain specific classes are placed in the plugin part of the system that extends the core classes. To support the Granite interface for CINET, software was developed with connections to memoization for domain, query, various graph library specific events (graph event, measure event, measure result event, parameter event), event handlers, and query parsers (graph query parser, measure query parser, and measure result query parser) plugins. GDSC system computes dynamics on a network. Memoization in this application involves with graphs, vertices, edges, vertex functions, initial state, and update scheme elements. Handling applications such as GDSC within CINET requires specific domain handlers such as domain, query, event, and event handler plugins.

6. Experimental Evaluations

In this section, empirical evaluations for the automated extensible memoization service are provided. Network science experimentation can achieve improved system-scale efficiency through this service.

6.1. Experimental Setup

Dataset: The experiment used 103 graphs and 109 measures available in CINET.

Network graphs were divided into small, medium, and large categories. The size of a graph $G = (V, E)$ is defined as $\mid G \mid = \mid V \mid + \mid E \mid$. A small graph is defined as $\mid G \mid < 100,000$, medium as $100,000 \leqslant \mid G \mid < 10,000,000$, and large as $\mid G \mid \geqslant 10,000,000$. The following three are respective examples of the three sizes of graphs, RND-G(n,p) Random Graph 1 (nodes:1,000, edges:4,971), RND-G(n,p) Random Graph 500 (nodes: 500,000, edges: 5.00E+06), and Seattle contact network (nodes: 3,207,037, and edges: 8.66E+07).

Galib is a scalable Graph Algorithm Library written in C++. Galib consists of more than 60 parallel and sequential graph algorithms. Galib is capable of handling large graphs up to 100 million nodes for sequential algorithms and 2 billion nodes for parallel algorithms. Galib employs techniques including streaming, external memory algorithms, and sampling based approximations [24]. NetworkX is an open source tool developed at Los Alamos National Laboratory. Compared to Galib, NetworkX doesn't perform well for large graphs but includes several useful features. It provides hundreds of graph algorithms [30]. CINET also incorporates eleven measures from the Stanford Network Analysis Project (SNAP) tool developed at Stanford University. These graph libraries

contain common algorithms; all of these are useful because of different features provided by different libraries [31].

Evaluation criteria: The objective of the memoization service is to improve the execution time in compute-intensive research. Therefore, with and without memoization execution times of an experiment are reported. Smaller running times after memoization are indicative of better performance, suggesting even a time consuming experiment will yield a prompt return of results to users.

Machine Configuration: Experiments are performed on Shadowfax, an HPC resource available at the Virginia Bioinformatics Institute (VBI). Shdowfax contains 912 processor cores, 5.4 TB of RAM, 40Gb/s InfiniBand network, 80TB parallel storage, 16 nVidia Tesla GPGPUs (7168 CUDA cores), and 3 FPGA based Convey HC-1 systems.

6.2. Results

CINET provides 63 Galib, 35 NetworkX, and 11 SNAP measures. The measures were executed with available small, medium, and large graphs (103 graphs). Tables 1, 2, and 3 report the top execution timings of a measure on three types of graphs (small, medium and large) without memoization and with memoization service. With the memoization service, existing experiments occured at least once in any earlier session. Thus, experiment requests can be reused from the outputs of previous experiments identified within the repository. In this case, the execution time with memoization service is comprised of the time required for querying and retrieving the result from the repository. Therefore, the execution times with memoization service in Table 1, 2, and 3 are less than the time required without memoization. For many types of experiment results, querying and retrieving times will be similar. This time may vary based on the number of records currently available in the RDF triplestore. At the time of this experiment 4,281 records are available in the RDF triplestore. Usage patterns by researchers was not investigated.

Table 1. Selected Galib measures runtime information

Measure	**Runtime (Sec) without Memoization**			**Runtime (Sec) with memoization**		
	Small	**Medium**	**Large**	**Small**	**Medium**	**Large**
1. Average shortest path distance	877	135,840	> 360,000	0.64	0.64	0.64
2. Generate a complete graph	240	25,200	> 151,200			
3. Shortest path distribution	23	18,000	> 43,200			

These results show that memoization services speed-up system throughput in large-scale research systems. For example, Table 2 shows "closeness centrality" takes more than a week (>720,000) to run on a large graph. The memoization facility takes only 0.64 second to produce the result from archives. Also from Table 1, 2, and 3, observe that the memoization service has a more profound impact on large and medium graph experiments than small graphs. Without memoization, running times for different graph types exponentially increase with graph size, but with memoization running time remains linear to the collection's metadata index.

Table 2. Selected NetworkX measures runtime information

Measure	**Runtime (sec) without Memoization**			**Runtime (sec) with memoization**		
	Small	**Medium**	**Large**	**Small**	**Medium**	**Large**
1. Compute closeness centrality	18,780	> 720,000	> 720,000	0.64	0.64	0.64
2. Check if a graph is biconnected	16.4	517	88,320			
3. Generate hypercube graph	60	3,120	64,800			

Graph and measure experiments identified the top time consuming experiment statistics and the less time consuming experiments. The above mentioned Tables (1, 2, 3) show that the top three time consuming measures are average shortest path distance (Galib), closeness centrality (NetworkX), and betweenness centrality (SNAP). From this experiment, the bottom three time consuming measures are: generate cycle graph (Galib), generate star graph (NetworkX), and degree centrality (SNAP). Figure 2 shows that memoization services take more time to return results than the smallest Galib measure running times. Out of 63 Galib measures, only 8 measures satisfy this criteria. On the other hand NetworkX (Figure 3) and SNAP (Figure 4) measures always provide

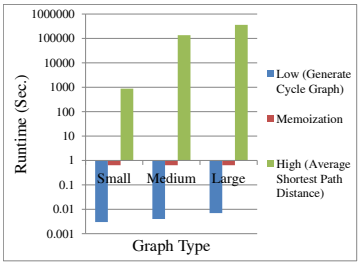

Fig. 2. Comparison of memoization with smallest and largest Galib measure running time.

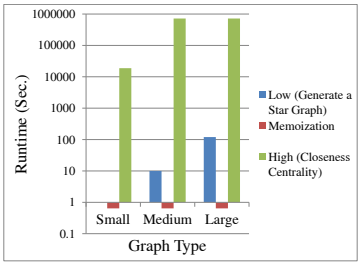

Fig. 3. Comparison of memoization with smallest and largest NetworkX measure running time.

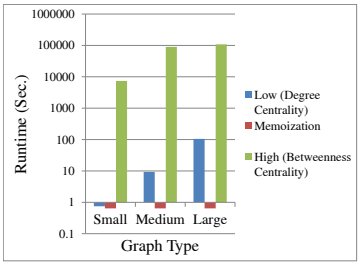

Fig. 4. Comparison of memoization with smallest and largest SNAP measure running time.

better efficiency with memoization service. These results demonstrate that memoization framework reduces the execution time for returning results in most cases.

Table 3. Selected SNAP measures runtime information

Measure	Runtime (sec) without Memoization			Runtime (sec) with memoization		
	Small	Medium	Large	Small	Medium	Large
1. Betweenness centrality	7,380	> 90,000	> 108,000	0.64	0.64	0.64
2. Closeness centrality	2,880	> 72,000	> 90,000			
3. Hubauth Centrality	11.97	151	1,598			

7. Summary and Future Work

In this paper, the design and implementation of an automated extensible memoization framework for network science were described. The framework's concepts, algorithms, and architecture were described. The memoization framework was implemented in the CINET framework case study, which is a CI based environment that supports computation and analytics. An experiment with diverse graphs and measures quantified the benefits of memoization. The experimental results show that the proposed extensible memoization framework increases the efficiency of the CI-based system. This type of system also provides numerous end-user functions. Fast response times empower teachers to show complex study designs and results in a classroom. Scientific experiments are time consuming and generate large simulation data. Memoization services assist scientists in fast retrieval of previously conducted, similar simulations. Researchers are also able to make comparisons between systematically managed collections of simulation results. This approach is extensible in supporting simulation applications from multiple network science domains without context-specific software development and integration. As a future work, we plan to implement higher-order search techniques within the DL service framework.

Acknowledgements

We thank all the reviewers and members of the Network Dynamics and Simulation Science Laboratory (NDSSL) for their comments and suggestions. This work has been partially supported by NSF NetSE Grant CNS-1011769, NSF SDCI Grant OCI-1032677, and DTRA CNIMS contract HDTRA1-11-D-0016-0001.

References

[1] F. Khalvati, H. Tizhoosh, M. Aagaard, Opposition-based window memoization for morphological algorithms, in: Computational Intelligence in Image and Signal Processing, 2007. CIISP 2007. IEEE Symposium on, 2007, pp. 425 –430. doi:10.1109/CIISP.2007.369207.

[2] T. Ikegaya, T. Tsumura, H. Matsuo, Y. Nakashima, A speed-up technique for an auto-memoization processor by collectively reusing continuous iterations, in: Networking and Computing (ICNC), 2010 First International Conference on, 2010, pp. 63 –70. doi:10.1109/ICNC.2010.46.
[3] Y. Liu, T. Teitelbaum, Caching intermediate results for program improvement, in: Proceedings of the 1995 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, ACM, 1995, pp. 190–201.
[4] W. Pugh, An improved replacement strategy for function caching, in: Proceedings of the 1988 ACM conference on LISP and functional programming, LFP '88, ACM, New York, NY, USA, 1988, pp. 269–276. doi:10.1145/62678.62719.
URL http://doi.acm.org/10.1145/62678.62719
[5] A. Heydon, R. Levin, Y. Yu, Caching function calls using precise dependencies, SIGPLAN Not. 35 (5) (2000) 311–320. doi:10.1145/358438.349341.
URL http://doi.acm.org/10.1145/358438.349341
[6] K. Swadi, W. Taha, O. Kiselyov, E. Pasalic, A monadic approach for avoiding code duplication when staging memoized functions, in: Proceedings of the 2006 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, PEPM '06, ACM, New York, NY, USA, 2006, pp. 160–169. doi:10.1145/1111542.1111570.
URL http://doi.acm.org/10.1145/1111542.1111570
[7] J. Mayfield, T. Finin, M. Hall, Using automatic memoization as a software engineering tool in real-world ai systems, in: Artificial Intelligence for Applications, 1995. Proceedings., 11th Conference on, IEEE, 1995, pp. 87–93.
[8] A. Da Costa, F. Franca, et al., The dynamic trace memoization reuse technique, in: Parallel Architectures and Compilation Techniques, 2000. Proceedings. International Conference on, IEEE, 2000, pp. 92–99.
[9] L. Ziarek, K. Sivaramakrishnan, S. Jagannathan, Partial memoization of concurrency and communication, SIGPLAN Not. 44 (9) (2009) 161–172. doi:10.1145/1631687.1596575.
URL http://doi.acm.org/10.1145/1631687.1596575
[10] P. Thiemann, Implementing memoization for partial evaluation, in: Programming Languages: Implementations, Logics, and Programs, Springer, 1996, pp. 198–212.
[11] C. Alvarez, J. Corbal, M. Valero, Fuzzy memoization for floating-point multimedia applications, Computers, IEEE Transactions on 54 (7) (2005) 922–927.
[12] T. Ishihara, F. Fallah, A way memoization technique for reducing power consumption of caches in application specific integrated processors, in: Proceedings of the conference on Design, Automation and Test in Europe-Volume 1, IEEE Computer Society, 2005, pp. 358–363.
[13] T. Ikegaya, T. Tsumura, H. Matsuo, Y. Nakashima, A speed-up technique for an auto-memoization processor by collectively reusing continuous iterations, in: Networking and Computing (ICNC), 2010 First International Conference on, IEEE, 2010, pp. 63–70.
[14] U. A. Acar, G. E. Blelloch, R. Harper, Selective memoization, SIGPLAN Not. 38 (1) (2003) 14–25. doi:10.1145/640128.604133.
URL http://doi.acm.org/10.1145/640128.604133
[15] M. Johnson, J. Dörre, Memoization of coroutined constraints, in: Proceedings of the 33rd annual meeting on Association for Computational Linguistics, Association for Computational Linguistics, 1995, pp. 100–107.
[16] I. Saha, R. Majumdar, Trigger memoization in self-triggered control, in: Proceedings of the tenth ACM international conference on Embedded software, ACM, 2012, pp. 103–112.
[17] E. Schnarr, J. Larus, Fast out-of-order processor simulation using memoization, in: ACM SIGOPS Operating Systems Review, Vol. 32, ACM, 1998, pp. 283–294.
[18] D. Borodin, B. Juurlink, Instruction precomputation with memoization for fault detection, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2010, pp. 1665–1668.
[19] C. Small, A functional approach to database updates, Information Systems 18 (8) (1993) 581–595.
[20] S. Chaudhuri, A. Aboulnaga, et al., Query optimization by sub-plan memoization, uS Patent 6,850,925 (Feb. 1 2005).
[21] T. H. Jordan, SCEC 2009 Annual Report, Southern California Earthquake Center.
URL http://www.scec.org/aboutscec/documents/SCEC2009_report.pdf
[22] C. L. Borgman, J. C. Wallis, M. S. Mayernik, A. Pepe, Drowning in data: digital library architecture to support scientific use of embedded sensor networks, in: Proc. JCDL 2007, 2007, pp. 269–277. doi:http://doi.acm.org/10.1145/1255175.1255228.
URL http://doi.acm.org/10.1145/1255175.1255228
[23] P. P. Leonardo Candela, Donatella Castelli, D4Science: an e-infrastructure for supporting virtual research, in: Proceedings of IRCDL 2009 - 5th Italian Research Conference on Digital Libraries, 2009, pp. 166–169.
[24] S. Abdelhamid, R. Alo, S. Arifuzzaman, P. Beckman, M. Bhuiyan, K. Bisset, E. Fox, G. Fox, K. Hall, S. Hasan, et al., Cinet: A cyberinfrastructure for network science.
[25] Granite website, http://ndssl.vbi.vt.edu/granite/.
[26] Gdsc website, http://ndssl.vbi.vt.edu/gdscalc/.
[27] S. Weibel, The dublin core: a simple content description model for electronic resources, Bulletin of the American Society for Information Science and Technology 24 (1) (2005) 9–11.
[28] A. Burks, D. Warren, J. Wright, An analysis of a logical machine using parenthesis-free notation, Mathematical tables and other aids to computation (1954) 53–57.
[29] M. Gonçalves, E. Fox, L. Watson, Towards a digital library theory: a formal digital library ontology, International Journal on Digital Libraries 8 (2) (2008) 91–114.
[30] A. A. Hagberg, D. A. Schult, P. J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Proceedings of the 7th Python in Science Conference (SciPy2008), Pasadena, CA, USA, 2008, pp. 11–15.
[31] J. Leskovec, Stanford Network Analysis Project, http://snap.stanford.edu/, [Online; accessed 17-July-2012] (2009).