Scott D. Stoller and Yanhong Liu

Stony Brook University

Resilience through distribution, replication, and coordination (ONR N000142112719)

Final Report

ACCOMPLISHMENTS

What were the major goals and objectives of this project?

Networked computers are running critical naval systems, for information infrastructure, decision support, cyber-physical system control, etc. These systems must be resilient under failures and attacks, such as Navy ship control systems facing offensives, or any critical software hit by mishaps.

Increasingly sophisticated software systems are built from a myriad of interconnected components providing a wide variety of functionalities. These components and functionalities have different degrees of importance under different scenarios. Resilience under failures and attacks requires the critical components for tolerating those failures and countering those attacks to continue to function.

This project proposes to develop a general framework for making systems resilient by automatically distributing system components among a collection of heterogeneous computing devices, replicating critical components on different devices, and coordinating distributed and replicated components to provide critical functionalities under failures and attacks.

The framework has two main facets: (1) building a model of failures and attacks, a model of critical components of the software system, and a model of capabilities of computing and communication devices; and (2) developing a method for determining optimal distribution and replication of critical functionalities, a method for constructing the overall distributed and replicated system, and a method for carrying out the critical functionalities for countering attacks and recovering from failures.

The overall framework aims to maximize resilience and minimize the effect of failures and attacks while satisfying the constraints of all computing and communication devices.

What was accomplished under these goals?

1. Overview

We developed a general framework for making systems resilient by automatically distributing software components among a collection of heterogeneous computing devices, replicating critical software components on multiple computing devices and using distributed algorithms to coordinate them.

The framework has two main parts: modeling and solving. We developed models of hardware failures, software components, and replication algorithms. A system configuration specifies which software components to run on each computing device, and, for each replicated software component, which replication algorithm to use to coordinate the replicas. We developed algorithms to synthesize resilient initial configurations and algorithms that determine a sequence of reconfiguration actions to take after failures and recoveries, to ensure that the system continues to provide critical functionality using the remaining resources. Specifically, we developed two solutions: one based on translation to Satisfiability Modulo Theories (SMT), and the other based on optimized search.

2. Modeling

The first challenge was to develop models that capture the characteristics of software components and replication algorithms that influence how a system should be configured to achieve the required resilience. In particular, formalizing the key characteristics that determine whether a software component requires replication and, if so, which replication algorithms are suitable, was challenging.

Failures. We first developed a model of failures capturing (1) the numbers of failures to tolerate for each type of hardware component (e.g., failures of 2 computers and 1 camera) and (2) the number of "simultaneous" failures to tolerate, i.e., the number of failures that occur in a time interval shorter than the system's reconfiguration time. A *failure scenario* is a set of failures of specific hardware components.

Software. We developed a model of software that captures critical functionalities, and key characteristics that affect replication, including whether the software component maintains essential volatile state, whether it maintains essential persistent state and if so whether it is small enough to be copied to new machines (new replicas) during reconfiguration, and whether the start-up time for a new instance of the software component is less than the acceptable downtime of the component's functionality during reconfiguration.

Our model also captures dependencies between software components, and whether a software component's functionality can be accessed only locally or also remotely. It captures as well the types of hardware devices (e.g., GPU, GPS, camera) used by the software; these devices may be standalone or integrated in a computer (e.g., a smartphone). Our model further captures computing platform requirements (instruction set architecture, OS) and computing resource requirements (number of cores, memory size).

Replication Algorithms. We developed a model of replication algorithms that focuses on key properties such as the quorum size needed to process requests, the quorum size needed for

membership changes (adding or removing replicas), active vs. passive replication (i.e., whether all replicas run the application, or one replica runs the application and disseminates the results to the others), handled failures, and synchrony assumptions.

Configurations and Reconfiguration. A system configuration specifies which software components to run on each computing device, and, for each replicated software component, which replication algorithm to use to coordinate the replicas. We developed a model of system reconfiguration after failures or recoveries. A system reconfigures by executing a sequence of *reconfiguration actions*, which include starting, stopping, and moving instances of software components, and changing the set of computers on which a software instance is replicated.

3. The Resilient Configuration Problem

We developed a formal definition of resilience. Essentially, a configuration *cfg* is *resilient* in a failure scenario *fs* if it currently provides all of the critical functionalities and, for each set of additional failures that can occur—i.e., each failure scenario *fs'* that is allowed by the failure model and is a superset of *fs*—the system can be reconfigured in *fs'* to a new configuration *cfg'* that provides all of the critical functionalities and is also resilient. Note that this is a recursive definition, with end cases corresponding to maximal failure scenarios according to the failure specification.

The *resilient configuration* problem is: given a failure model, and a system model including a list of critical functionalities, find a resilient configuration, if one exists. The *optimal resilient configuration* problem is: given a quality metric on the configurations, find a resilient configuration that has the highest quality.

4. Solution by Translation to SMT using Z3

Our first solution to the resilient configuration problem is based on translation to Satisfiability Modulo Theories (SMT). We developed a prototype implementation using Z3, a state-of-the-art SMT solver. Specifically, we wrote a Python program that generates a set of Z3 formulas that encode a given problem instance: the set of formulas is satisfiable if and only if a resilient configuration exists.

Z3, like other SMT solvers, is based on first-order logic and does not support quantification over sets or functions. Consequently, most of the information in our model must be encoded in a low-level way as collections of Boolean variables. This greatly complicates the implementation, and makes debugging very difficult, because the generated Z3 formulas are large, even for very small problem instances. For example, for a problem instance with only 3 computers, 2 software components, 2 devices, and 2 replication algorithms, the generated formulas, in s-expression format (which is more compact than the default format), total about 920 lines.

5. Solution by Optimized Search using DistAlgo

Our second solution to the resilient configuration problem generates sets of possible configurations and explicitly searches through them to find resilient configurations. We

developed a prototype implementation in DistAlgo, our extension of Python for high-level programming of distributed algorithms. DistAlgo supports high-level queries with patterns, including logic quantifications with patterns as queries. The implementation is much clearer than for Z3, because high-level representations, such as sets and functions, can be used directly. The DistAlgo representation of our models and the logical formulas defining resilience are similar to their mathematical formulations.

The DistAlgo prototype solves both the resilient configuration problem and the optimal resilient configuration problem. It incorporates several optimizations that drastically reduce the number of configurations that are generated and evaluated for resilience. This includes several pruning optimizations, which avoid enumeration of many configurations that are not resilient or would use more resources than necessary to achieve resilience. It also includes two powerful, specialized, symmetry-based reductions, which avoid separate evaluation of configurations that are symmetric variants of each other and hence are either all resilient or all not resilient.

6. Comparison of the prototypes

Completeness. The DistAlgo prototype is complete. The Z3 prototype is not complete, for the following reasons.

The Z3 prototype implements a non-recursive variant of the above notion of resilience, which requires that the system provides all critical functionality (instead of requiring that it is resilient) after reconfiguration in failure scenario fs'. The recursive definition cannot be directly represented in Z3's first-order logic, though it can be simulated using a loop in the Python driver program. Also, the Z3 implementation does not support finding optimal resilient configurations. Extending it to support this would require significant additional work; in contrast, extending the DistAlgo implementation to support this was straightforward.

Assurance. Assuring correctness of the DistAlgo implementation is much easier than for the Z3 implementation. An obvious reason is that it is smaller: the core (not including optimizations) is about 170 lines, compared with about 275 for the Z3 implementation. But the key reason is that *all of the formulas are explicitly present in the DistAlgo code* and are independent of the problem instance. In contrast, the Z3 implementation dynamically generates a large collection of formulas—over 900 lines, even for very simple examples—that are specific to the problem instance. Debugging the code, and assuring correctness of the generated formulas in all cases, is very challenging.

Scalability. The primary motivation for the Z3 implementation is its expected scalability. The Z3 implementation uses symbolic reasoning, and Z3 has a sophisticated reasoning engine. Thus, it is expected to be much faster than brute-force search for a resilient configuration.

As discussed above, the DistAlgo implementation exploits search to find resilient configurations, but incorporates several optimizations that drastically reduce the number of configurations that are generated and evaluated for resilience.

Our initial experiments, described under Evaluation below, show that the DistAlgo implementation already provides acceptable performance for small to medium size problem instances.

This short project leaves open two interesting directions for future work, to further improve scalability while still providing high assurance. One is to develop additional optimizations for the DistAlgo implementation. The other is to take advantage of the Z3 implementation's expected scalability by using it to find (candidate) resilient configurations, and use the DistAlgo implementation to check resilience of those configurations, to provide high assurance.

7. Evaluation

We developed a model of an autonomous driving system. The software components are based on the architecture of the open-source software Autoware. The software model includes about half a dozen components, including perception, vehicle localization, planning, control, and vehicle interface. Some of them maintain critical volatile state and require replication; others can simply be restarted on a surviving computer after a failure. The hardware model includes several sensors and multiple ARM-based embedded computers. The computing platform is based on QualComm's Snapdragon Ride, which uses multiple smaller embedded computers with ARM processors, for greater scalability and power-efficiency than a single large x86_64-based computer. The model also includes a Linux laptop and smartphone that are in the vehicle. Some software components have alternative (less preferred) versions that can run on these devices, in case some embedded computers fail.

We evaluated the DistAlgo implementation on a few variants of this system (we have not evaluated performance of the Z3 implementation, since it is not completely debugged, due to the challenges discussed in Section 3). We varied the number of embedded computers and the maximum number of failures, to assess scalability in this dimension. Our resilience analysis finishes within several minutes for systems with up to 5 embedded computers and up to 4 failures.

What opportunities for training and professional development has the project provided?

An M.S. student received training and professional development while working on this project during the current reporting period. The student studied and compared existing approaches to reconfiguration of replication algorithms, studied and evaluated a few candidate constraint solvers, and helped implement the prototypes.

How were the results disseminated to communities of interest?

We have written a detailed, 30-page description of the framework, the prototypes, and example system models and analysis results. The document includes extensive discussion of design rationale and potential extensions. We plan to write a paper describing the results, publish it on arXiv.org, and submit it for publication in a refereed conference or journal.

The publications listed in the Products section describe work that, although done primarily under the auspices of other projects, is related to our work on this project. The papers on aggregation and on programming with rules and everything else are related to extending DistAlgo with additional features that would benefit the use of DistAlgo in this project. The paper on Black-Box Simplex architecture describes a runtime-assurance technique for increasing resilience.

Honors and Awards

Nothing to report.

Technology Transfer

Nothing to report.

PARTICIPANTS & OTHER COLLABORATING ORGANIZATIONS

What individuals have worked on this project?

Scott D. Stoller
Project Role: PD/PI
National Academy Member: no
Months Worked: 2
Foreign Collaboration: none
Foreign Travel: none

Yanhong Liu
Project Role: co PD/PI
National Academy Member: no
Months Worked: 1
Foreign Collaboration: none
Foreign Travel: none

Balaji Jayasankar
Project Role: Graduate Student (research assistant)
National Academy Member: no
Months Worked: 1
Foreign Collaboration: none
Foreign Travel: none

Have other collaborators or contacts been involved?

Nothing to report.

What other organizations have been involved as partners?

Nothing to report.

Students

Number of students receiving STEM degrees during the reporting period: 0
Number of undergraduate and graduate STEM participants during the reporting period: 1

PRODUCTS

Journal Publications

Yanhong A. Liu and Scott D. Stoller. Recursive Rules with Aggregation: A Simple Unified Semantics. Journal of Logic and Computation.
Status: awaiting publication
Acknowledges federal support: yes

Conference Publications

Yanhong A. Liu and Scott D. Stoller. Recursive Rules With Aggregation: A Simple Unified Semantics. *Logical Foundations of Computer Science (LFCS 2022)*, volume 13137 of Lecture Notes in Computer Science, pages 156–179. Springer-Verlag, 2021.
Status: published.
DOI: https://doi.org/10.1007/978-3-030-93100-1_11
Acknowledges federal support: yes

Usama Mehmood, Sanaz Sheikhi, Stanley Bak, Scott A. Smolka, and Scott D. Stoller. The Black-Box Simplex Architecture for Runtime Assurance of Autonomous CPS. 14th NASA Formal Methods Symposium (NFM 2022), volume 13260 of Lecture Notes in Computer Science, pages 231-250. Springer-Verlag, 2022.
Status: published.
DOI: https://doi.org/10.1007/978-3-031-06773-0_12
Acknowledges federal support: yes

Yanhong A. Liu, Scott D. Stoller, Yi Tong, and Bo Lin. Integrating Logic Rules with Everything Else, Seamlessly. The 25th International Symposium on Practical Aspects of Declarative Languages (PADL 2023).
Status: under review.
Acknowledges federal support: yes

Yanhong A. Liu, Scott D. Stoller, Yi Tong, and K. Tuncay Tekle. Benchmarking for Integrating Logic Rules with Everything Else. The 25th International Symposium on Practical Aspects of Declarative Languages (PADL 2023).
Status: under review.
Acknowledges federal support: yes

Books Or Other Non-periodical, One-time Publications

Nothing to report.

Other Publications

Liu, Yanhong A. and Stoller, Scott D. Recursive Rules with Aggregation: A Simple Unified Semantics, arXiv:2007.13053 [cs.DB], July 2020 (updated September 2022). https://arxiv.org/abs/2007.13053.
Status: published.

Liu, Yanhong A. and Stoller, Scott D. and Tong, Yi and Lin, Bo and Tekle, K. Tuncay. *Programming with Rules and Everything Else, Seamlessly*. arXiv:2205.15204 [cs.PL], May 2022. https://arxiv.org/abs/2205.15204.
Status: published.

Websites

Nothing to report.

Other Products

Nothing to report.

CHANGES/PROBLEMS

Nothing to report.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
20221028	Final Report	20210809	20220808

4. TITLE AND SUBTITLE

Resilience through Distribution, Replication, and Coordination

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	N000142112719	

5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER

6. AUTHOR(S)

Stoller, Scott D.
Liu, Yanhong

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Research Foundation for The State University of New York
West 5510 Frank Melville Memorial Library
Stony Brook, NY 11794-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
US Navy Office of Naval Research	ONR	

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Networked computers run many critical systems, including naval systems. These increasingly sophisticated systems are built from heterogeneous hardware components and a myriad of interdependent software components. These systems must be resilient under failures and attacks.

In this project, we developed a general framework for making systems resilient by automatically distributing software components among a collection of heterogeneous computing devices, replicating critical software components on multiple computing devices and using distributed algorithms to coordinate them.

The framework has two main parts: modeling and solving. We developed models of hardware components, software components, replication algorithms, and failures. A system configuration specifies which software components to run on each computing device, and, for each replicated software component, which replication algorithm to use to coordinate the replicas. We developed algorithms to synthesize resilient initial configurations and algorithms that determine a sequence of reconfiguration actions to take after failures and recoveries, to ensure that the system continues to provide critical functionality using the remaining resources.

We implemented the models and algorithms in a prototype and applied it to models of an autonomous driving system. The experimental results demonstrate the framework's ability to flexibly reconfigure complex systems in order to achieve specified resilience goals.

15. SUBJECT TERMS
fault-tolerant computer systems, resilient computer systems, distributed systems, replication

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	UU	

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Scott D. Stoller	631-632-1627

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE.
Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE.
State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED.
Indicate the time during which the work was performed and the report was written.

4. TITLE.
Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER.
Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER.
Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER.
Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER.
Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER.
Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER.
Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/ FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

PREVIOUS EDITION IS OBSOLETE.

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18