UNIVERSITY OF NORTH CAROLINA CHARLOTTE[1]

Lawrence Livermore National Laboratory[2]

RDS: A Cloud-Based Metaservice for Detecting Data Races in Parallel Programs

14th IEEE/ACM International Conference on Utility and Cloud Computing, Dec. 8th, 2021

Yaying Shi[1], Anjia Wang[1], Yonghong Yan[1] and Chunhua Liao[2]

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. LLNL-PRES-829787
Lawrence Livermore National Security, LLC This work is also supported by the National Science Foundation under the award 2015254

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

Introduction

- What is Data Race in parallel program?

```
#pragma omp parallel for
    for (i=0; i< len-1; i++)
        a[i] = a[i+1]+1;
```

Thread 1
a[1] = a[2] + 1
a[2] = a[3] + 1

Thread 2
a[3] = a[4] + 1
a[4] = a[5] + 1

Data race may crash of program execution, corrupted results.

Data races are hard to detect, reproduce, locate, and eliminate.

Data Race Detection Tools

- Static

  Use static program analysis techniques

  1. limited due to unknown program semantics at compile-time
  2. difficult for the tools to enumerate all possible execution scenarios by static analysis

- Dynamic

  Rely on dynamic race detection techniques by collecting and analyzing memory access information

  Tools: Intel Inspector , Archer , ThreadSanitizer, and ROMP

Challenges for Single Data Race Detection Tool

- It is hard to install and configure a tool.
- Tools have very different user interfaces and command line options.
- Each tool has its own limitations of detecting races of different types and behaviors .

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

Motivation

- RaceDetectionService (RDS) integrates four dynamic race detection tools and provides a convenient cloud-based service for users to detect data races of parallel programs.
- RDS provides a generic, two-level composable service framework that can be used for combining multiple tools
- A unified set of language-neutral REST API functions and input/output data formats in JSON have been developed for the communication between different layers of RDS.
- We propose and compare a set of policies that the metaservice uses to merge potentially conflicting results from multiple data race detection tools.

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

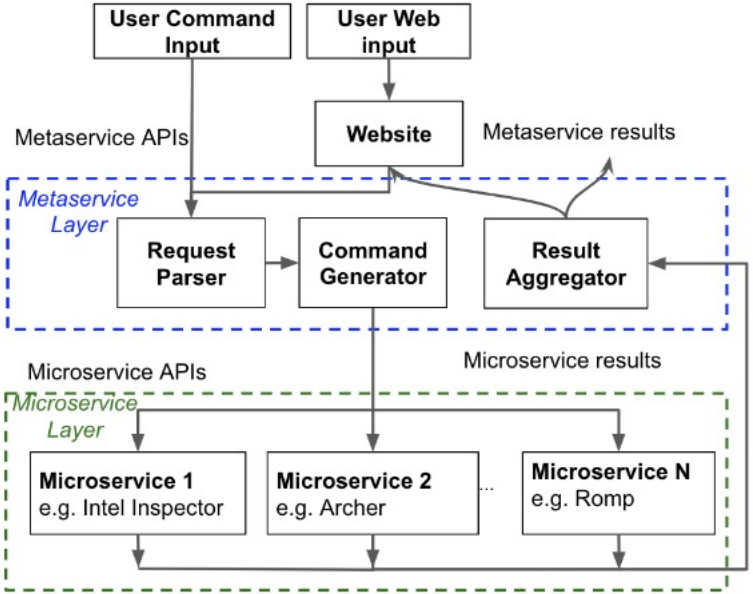
Design and Implementation

Service-Oriented Architecture (SOA)

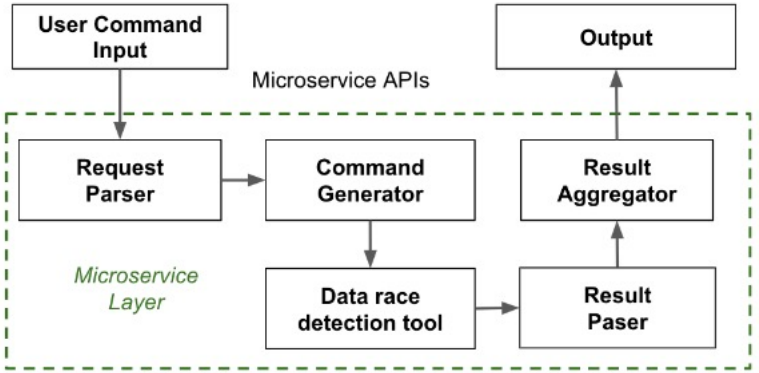
- A metaservice layer processes a user's request, generates new service requests with a set of commands for each microservice tool, then sends the service requests to the corresponding microservices.

- A microservice layer encapsulates individual race detection tools process the requests from the metaservice and respond to the metaservice with the service results.

Architecture



Design of RaceDetectionService (RDS)



Design of a microservice

RESTful API Design

HTTP Method	URL	Parameters	Description
GET	/RDS	N/A	List available race detection services' IDs, such as meta, micro-archer, micro-romp, etc.
POST	/RDS/service-id	SyncFlag, file, options	Send a file to a race detection service specified by its id, with extra options. The service will finish all the work and return a report if the synchronous flag is set to true. Otherwise, the service will immediately return a request ID and a key to authenticate possible HTTP DELETE requests. The actual race detection work will be executed in the background.
GET	/requests	N/A	List all requests submitted to all services
GET	/requests/request-id	N/A	Check the status of a specific request, return a status of nonexist, finished, pending, running.
DELETE	/requests/request-id	key	Cancel an ongoing request, return a status of nonexist, success or failure.

JSON-based Data Exchange Format

```
{
  "program": "a.out",
  "data_races": [
    {
      "read": {
        "location": ["file1.c",64,12],
        "symbol": "A[i]"
      },
      "write": {
        "location": ["file1.c",64,11],
        "symbol": "A[i]"
      },
      "microservices": [
        {"Archer": true},
        {"ROMP" : true},
        {"ThreadSanitizer": true},
        {"Inspector": false}
       ]
    },
    {
      "read": {
        "location": ["file2.c",132,7],
        "symbol": "b"
      },
      "write": {
        "location": ["file2.c",246,31],
        "symbol": "b"
      },
      "microservices": [
        {"Archer": true},
        {"ROMP" : false},
        {"ThreadSanitizer": true},
        {"Inspector": true}
      ],
    }
  ],
  "raw_output": [
        {"Archer": "shorturl.at/uzJR7"},
        {"ROMP" : "shorturl.at/enwH4"},
        {"ThreadSanitizer": "shorturl.at/otO67"},
        {"Inspector": "shorturl.at/dH389"}
  ],
  "aggregate_policy": "Union"
}
```

Result Aggregation Policies

- Set-union
- Set-intersection
- Majority voting
- Random selection
- Weighted voting
- OpenMP-construct specific weighted voting

Directives	Archer	ThreadSanitizer	Intel Inspector	ROMP
Parallel	1.00	0.93	0.92	1.00
Parallel for	0.92	0.77	0.72	0.98
Parallel section	1.00	0.67	0.67	0.67
task	0.88	0.36	0.78	0.78
task loop	0.67	0.67	1.00	1.00
simd	0.50	0.50	0.50	0.50
threadprivate	1.00	0.57	0.67	0.60
master	1.00	1.00	1.00	1.00
target	0.50	0.80	0.33	0.80
flush	1.00	1.00	1.00	1.00
single	1.00	1.00	1.00	1.00
atomic	1.00	0.00	1.00	1.00

Directive-specific F1-score of the Tools

Security and Scalability

- Security
  Rootless Docker in docker
- Scalability
  Cloud platform

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

Evaluation

Tool Result	Ground Truth		Recall	Specificity	Precision	Accuracy	F1 Score
	True	False					
True	TP	FP	TP / (TP + FN)	TN / (TN + FP)	TP / (TP + FP)	(TP+TN) / (TP + FP + TN+ FN)	2 * (P * R) / (P + R)
False	FN	TN					

Metrics for Evaluation

Tool	Languages	Version	Compiler	Flag(s)/Environment var
Archer	C/C++	release_60	Clang/LLVM 6.0	export TSAN_OPTIONS="ignore_noninstrumented_modules=1"; -larcher -fopenmp
Intel Inspector		2020(build 603904)	Intel Compiler 19.1.0.166	-collect ti3 -knob scope=extreme -knob stack-depth=16 -knob use-maximum-resources=true -fopenmp
ROMP		20ac93c	GCC 7.4	-g -std=c++11 -fopenmp -lomp
ThreadSanitizer		10.0	Clang/LLVM 10.0	export TSAN_OPTIONS="ignore_noninstrumented_modules=1"; -fopenmp -fsanitize=thread

Selected Data Race Detection Tools

Individual Data Race Detection Tools vs. RDS

Tool	Aggregate Policy	TP	FP	TN	FN	Recall	Specificity	Precision	Accuracy	TSR	Adjusted F1
Intel Inspector	Union	126	10	144	36	0.778	0.935	0.926	0.854	0.935	**0.791**
ThreadSanitizer		117	**1**	**151**	29	0.801	**0.993**	**0.991**	**0.899**	0.882	0.781
Archer		116	**1**	143	29	0.800	**0.993**	**0.991**	0.896	0.855	0.757
ROMP		**133**	52	106	**26**	**0.836**	0.671	0.719	0.754	**0.938**	0.725
RDS	Union	**144**	71	97	**19**	**0.883**	0.577	0.700	0.728	**0.979**	0.746
	Intersection	115	**3**	**165**	48	0.706	**0.982**	0.975	0.846	**0.979**	0.802
	Random	133	20	148	30	0.816	0.881	0.869	0.849	**0.979**	0.824
	Majority (Positive tie breaker)	132	9	159	31	0.810	0.946	0.879	0.868	**0.979**	0.850
	Majority (Negative tie breaker)	126	**3**	**165**	37	0.773	**0.982**	0.977	0.875	**0.979**	0.845
	Weighted Vote	132	8	160	31	0.810	0.952	0.943	0.882	**0.979**	0.853
	Directive-Specific Weighted Vote	130	**3**	**165**	33	0.798	**0.982**	**0.977**	**0.891**	**0.979**	**0.860**

* TSR refers to Tool support rate. Adj. F1 = F1 * TSR

Multiple-file packages of NAS Parallel Benchmarks

Tool	Aggregate Policy	TP	FP	TN	FN	Accuracy	TSR	Adj. Accuracy
Intel Inspector	Union	0	100	630	0	0.863	**100%**	0.863
Thread Sanitizer		**0**	0	**649**	0	**1.000**	88.9%	0.889
Archer		0	1	630	0	**0.998**	**100%**	**0.998**
ROMP		0	**0**	151	0	**1.000**	20.6%	0.210
RDS	Union	0	100	630	0	0.863	100%	0.863
	Intersection	0	**1**	**729**	0	**0.998**	100%	**0.998**
	Random	0	22	708	0	0.970	100%	0.970
	Majority (Positive tie breaker)	0	**1**	**729**	0	**0.998**	100%	**0.998**
	Majority (Negative tie breaker)	0	11	719	0	0.985	100%	0.985
	Weighted Vote	0	**1**	**729**	0	**0.998**	100%	**0.998**

* TSR refers to Tool support rate. Adj. Accuracy = Accuracy * TSR

Overhead of RDS

- Time overhead
- Storage overhead

Tools	Microservice	Native Use	Overhead	Extra Storage
Intel Inspector	15205 s	15129 s (99.5%)	76 s (0.50%)	90MB
ThreadSanitizer	13086 s	13015 s (99.45%)	71 s (0.55%)	90MB
Archer	1533 s	1467 s (95.46%)	66 s (4.54%)	90MB
ROMP	1034 s	966 s (92.95%)	68 s (7.05%)	90MB

Final total storage overhead for RDS is about 598MB, which is 1.84% of the total docker image size of 32.58 GB.

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

Conclusion

- RDS is the first effort in the HPC community to create cloud-based metaservice tools that consolidate other tools.
- Its architecture can be used to create metaservice for compilers, performance tools, auto-tuning tools, etc..
- we demonstrate that policies incorporated with fine-granularity domain information such as OpenMP directive types deliver more accurate results than standard policies in our experiments.

Agenda

- Introduction
- Motivation
- Design and Implementation
- Evaluation
- Conclusion
- Future work

Future Work

- Compose supportive program analyses from different compilers and tools
- Support package management systems such as SPACK

Thank You

Q&A

Prepared by Lawrence Livermore National Laboratory under Con-tract DE-AC52-07NA27344 (LLNL-CONF-809187) and supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. This work is also supported by the National Science Foundation under the award 2015254