AD for pap421: AppSysFusion: Integrating Application and System Data for Execution-Time Diagnosis of HPC Application Performance Variation

Summarize the experiments reported in the paper and how they were run. Please list the experiments reported in this paper and the platform(s) utilized, including system details and all the software versions employed. MathJax is enabled so you can enter LaTeX mathematical notation within.

[We will refer to this section as (1)]

We have architected and deployed an end-to-end capability for collection of synchronized system monitoring data and event-driven application performance/progress data that are both timestamped using node-local system clocks. We also describe our storage, analysis and visualization of that data.

The below demonstrations and experiments were utilized to convey the scope and impact of the work as well as the lack of adverse impact on application performance.

- Demonstrations that we are collecting, transporting, and storing application progress/performance data at reasonable scale:
  - We demonstrate use of our implementation at a scale of 290 nodes and 1200 MPI ranks on a 1488 node/47616 core production HPC system at Sandia National Laboratories.
  - There were 6 such application runs which lasted for approximately one week each.
  - The LDMS deployment implemented a fan-in ratio of 93 sampler daemons (1 daemon per compute node) per aggregator daemon.
- Experiments to quantify overhead of sampling and writing application progress/performance data to the LDMS streams interface:
  - The application tested was EMPIRE and the simulation was electron migration. The simulation was run on 600 MPI ranks across 300 nodes of Sandia's 1488 node Eclipse system and lasted approximately 40 minutes.
  - Performed 32 independent experiments which were identical in input deck, binaries, and host compute nodes.
  - Four different application monitoring configurations were utilized with a round-robin ordering:
    - 8 baseline experiments where the application neither sampled Kokkos kernel timings nor connected to an LDMS daemon for transmission of data
    - 8 experiments where each rank made a connection to a node-local LDMS daemon and timings of 0.1% of Kokkos kernel executions were collected and published to the LDMS Streams interface
    - 8 experiments where each rank made a connection to a node-local LDMS daemon and timings of 1.0% of Kokkos kernel executions were collected and published to the LDMS Streams interface
    - 8 experiments where each rank made a connection to a node-local LDMS daemon and timings of 10% of Kokkos kernel executions were collected and published to the LDMS Streams interface

    - Within the SteamsOverheadTesting artifact directory are files that contain the raw results of the runtimes for the various sampling percentages. There were two outlier datapoints in the results which had an increased run time of 25\%. One was in a baseline run and another was in a 0.1\% sampling run. We removed these two datapoints from our overhead calculations as they appear to be caused by external factors (e.g., contention in network or filesystem) and are not representative of normal operation of their respective groups.
    - Our results are based on averaging the group datapoints and calculating the mean additional application execution time for each group over the mean of the baseline group.
- Experiments to identify anomalous (outlier) application performance in a production HPC environment:
    - The application tested was EMPIRE and the simulation was electron migration. The simulation was run on 32 MPI ranks across 4 nodes of Sandia's 1488 node Eclipse system.
    - Experiments were run and were identical in input deck, application binaries, and Kokkos kernel performance sampling rate.
        - Starting from the same restart file (used restart file in order to run the simulation in an area where significant physics was being performed).
    - Using the production SLURM batch scheduling system EMPIRE simulation jobs were submitted daily over a two-month period.
    - Both system and application data collected as part of these experiments is provided in (2).
        - Within the Data artifact directory is a file with the execution time data for the 52 successful EMPIRE runs described in section V-B in the paper
        - Within the Data artifact directory are files that contain the execution times and collected application and system data for the 1 baseline and 3 anomalous EMPIRE runs described in section V-C in the paper
    - A description of how we analyzed this data to discover the characteristics used to discriminate between normal and outlier application runs is described below.

Functionality of our system service includes analysis and use of our dashboard for data visualization. The monitoring cluster (described in (3)) was used for all work in the paper. Location and description of code and scripts developed for the analysis and visualization are in the artifacts list.

- Demonstration of utility of the visualization and analysis was performed in the exploration of application and system metrics in a "baseline" (normal duration) run and a comparison "outlier" (~36% longer duration) run (Section V-C, with experiments described above). The visualization and analysis were invoked manually. Screenshots of the visualizations are in the paper.
- Functionality and utility demonstration listed in the paper is the production use of these components from the July 2021-onward time frame of deployment and includes six 290-node 4-ranks/node runs of EMPIRE each of which ran for 1 week and thirty-two 300-node 2 ranks/node overhead experiments described above.

- An experiment to assess Grafana query latency and analysis module performance is reported in the paper (Section VI-B). The parallel database was configured as 16 independent databases with 74 TB of production system+application data collected over about 6 weeks. We timed the Grafana visualization response, which includes the time it takes to perform the data query, the time of the underlying analysis module computation, along with the time taken to format the pandas DataFrame into a JSON object and the time for Grafana to plot the JSON object. The chosen analysis module, called singleMetricRateTSDSOS which is included in the artifacts, queries the DSOS database for a given metric and finds the difference between successive points in time per node ID using pandas groupby and diff functions. Queries for each of four time ranges (3 hour, 12 hour, 24 hour, and 48 hour) for a single metric (Active memory) for a single node were performed. Results are shown in the paper.

Platform and software information is supplied along with the build and configure information in (3) This work leverages and extends existing tools with new capabilities enabled by the configure information.

==

List for every author artifact

Persistent ID (DOI, GitHub URL, etc.)

Artifact name

Citation of artifact (if known)

[We will refer to this section as (2)]

Artifacts that have been contributed to an existing open-source code base are listed below:

- https://github.com/ovis-hpc/ovis/tree/OVIS-4/kokkosConnector
- Kokkos-connector file (addition to existing LDMS)
- No separate citation for the kokkos-connector itself. Citation of LDMS is in paper [4] as well as the URL https://github.com/ovis-hpc/ovis

- https://github.com/ovis-hpc/ovis/blob/OVIS-4/ldms/src/ldmsd/ldmsd_stream.c and related files in the ldmsd directory.
- Streams enhancements long-lived connections and large message sizes, etc (addition to existing LDMS)
- No citation for the streams enhancements themselves. Citation for LDMS is as above.

- https://github.com/ovis-hpc/ovis/blob/OVIS-4/ldms/src/store/kokkos_appmon/kokkos_appmon.c
- LDMS plugin to store kokkos data to SOS and CSV format (addition to existing LDMS)
- No citation for the stores. Citation for LDMS is as above.

- https://github.com/vsurjadidjaja/kokkos-tools/tree/patch-1/common/kokkos-sampler
- kokkos-sampler (addition to existing kokkos tools)
- No citation for the kokkos-sampler. Citations for Kokkos are in paper [2] and [3] as well as the URL https://github.com/kokkos (multiple repositories)

- https://github.com/ovis-hpc/sos/tree/SOS-5/rpc and https://github.com/ovis-hpc/sos/tree/SOS-5/libtirpc
- Modified open source rpc implementation to support concurrent client execution. Code spread across a number of files that was contributed to SOS/DSOS enabling parallel support in execution of this work. Providing which lines were contributed to this is beyond the scope of this document.
- No citation for the changes. Citation for SOS is in the paper [6]

- https://github.com/nick-enoent/numsos, https://github.com/nick-enoent/sosdb-grafana, https://github.com/nick-enoent/dsosds
- HWS: files that enable connection of Grafana to our analysis infrastructure. numsos is the suite of python modules that handles analysis of data provided by sosdb-grafana. sosdb-grafana is backend software that handles httpd requests and queries DSOS containers for the requested information. dsosds is a third party datasource plugin for grafana that handles and configures grafana requests for the sosdb-backend
- No citation for the changes. Citation for HWS is in the paper [5]

Other artifacts or information necessary to understand environment and experiments:

- https://github.com/ovis-hpc/SC22Artifacts/tree/main/AnalysisViz
- Code and scripts related to analysis and visualization
- N/A

- https://github.com/ovis-hpc/SC22Artifacts/tree/main/SystemInfo
- System information output files from collect_environment.sh
- N/A

- https://github.com/ovis-hpc/SC22Artifacts/tree/main/Data
- Application-System data and code run durations
- N/A

- https://github.com/ovis-hpc/SC22Artifacts/tree/main/StreamsOverheadTesting
- Code run durations for various kernel sampling rates
- /N/A

Experimental setup:

Give details on the experimental environment.

These items were used in the experiments, but not created or changed by the author.

Fill in whatever is relevant to your paper and leave the rest blank.

Have you provided container image(s) link(s) in the 'Author artifacts' section (or equivalent alternative methodology that enable an easy reproducibility process)?}

Please, justify why it is impossible to provide a container (or equivalent) and, next, detail how your artifact will be reproduced. Details on how to reproduce the artifact must include how to install, compile and execute the provided artifacts and need to be provided in Phase 1, in the Artifact Description. Authors need to specify all the software dependencies, how to solve any version conflict that reviewers might face, and any other information essential to fully reproduce the artifact. Failure to provide clear and complete documentation on how to fully test the artifact will result in the impossibility of assigning the associated badges.

[We will refer to this section as (3)]

We cannot produce a container comprising all of the artifacts primarily because the (1) application used (EMPIRE) and its input decks are export-controlled and, therefore, not publicly releasable, and (2) we run the primary data collection tool (LDMS) on bare metal in continuous mode, therefore, a container is not useful.

LDMS, Kokkos are open-source tools, whose basic capabilities are already established, which are leveraged for this work. We provide URLS to artifacts in within their source trees (whose repositories include code and build instructions) in the prior section, and LDMS and sampler configuration details in this section, so one could install and configure the augmented tools on any system. The URLS all point to the appropriate version to use for reproducibility.

Reproduction of the experimental results would require a deployment on an identical system and running the EMPIRE with an identical workload. Since that would not be possible, we have provided source code for the new functionalities developed in this work that can be examined. We have also provided data from the experiments described in the paper as artifacts.

Application EMPIRE:

EMPIRE cannot be provided. The code is built upon Kokkos. Kokkos can be obtained in the URL in (2).

Details of HPC compute system where application and system data was collected:

Eclipse, our 1.3 Petaflop production HPC system, can be summarized as follows:

- 1488 compute nodes, 12 login nodes, 8 administrative nodes, 24 Lustre gateway nodes.
- All nodes have 128 GB RAM, 2 Intel Broadwell processors (E5-2695 v4 @ 2.10GHz), 1 gigabit Ethernet, 100 gigabit Omnipath.
- Lustre gateway nodes additionally have Mellanox ConnectX-4 Infiniband links to storage servers.
- Administrative nodes additionally have 2 TB local flash storage and 10 gigabit Ethernet.

Output of the collect_environment.sh script, as modified to match local hardware and data-release policy constraints, is in the artifacts.

Details of Monitoring cluster with LDMS store plugin and storage, analysis, and backend visualization server:

Our monitoring cluster can be summarized as follows:

- 16 storage nodes with 1.5 TB RAM, 47 TB NVME raid storage (composed of 8 Intel DC P4610 7.6 TB drives), 2 Intel Cascade Lake processors (Gold 6240R @ 2.4GHz), 10 gigabit Ethernet, 100 gigabit Infiniband
- 1 login node with 1.5 TB RAM, 2 Intel Cascade Lake processors (Gold 6240R @ 2.4GHz), 10 gigabit Ethernet, 100 gigabit Infiniband

Output of the collect_environment.sh script, as modified to match local hardware and data-release policy constraints, is in the artifacts.

Building kokkos-sampler on the HPC system:

The kokkos-sampler can be built with the provided Makefile. It is important to note that Kokkos must already be installed alongside the application in use. More information can be found at https://github.com/kokkos.

Building LDMS on the HPC system:

LDMS was used with the additional artifacts and can be obtained at the URL in (2). A generic compute node build of LDMS (without authentication) for reproducibility purposes can be configured as below where the target install directory in this example is /opt/ovis:

\begin{verbatim}

```
../configure CFLAGS=-ggdb3 -O0 -Wall -Werror --prefix=/opt/ovis --libdir=/opt/ovis/lib64 --
libexecdir=/opt/ovis/lib64
```

\end{verbatim}

Building Kokkos-connector on the HPC system:

Use the following Makefile code to build and link the Kokkos connector with the Kokkos sampler:

```
OVIS_DIR=/projects/ovis/streams/opt/ovis

CXX=g++

CXXFLAGS=-O3 -std=c++11 -g \

-I$(OVIS_DIR)/include/ -I./include

SHARED_CXXFLAGS=-shared -fPIC

LDFLAGS=-L$(OVIS_DIR)/lib

LIBS=-lldmsd_stream -lldms -lrt

 all: kp_kernel_ldms.so

 MAKEFILE_PATH := $(subst Makefile,,$(abspath $(lastword $(MAKEFILE_LIST))))

 CXXFLAGS+=-I${MAKEFILE_PATH}

 kp_kernel_ldms.so: ${MAKEFILE_PATH}kp_kernel_ldms.cpp
${MAKEFILE_PATH}kp_kernel_info.h $(CXX) $(SHARED_CXXFLAGS) $(CXXFLAGS) $(LDFLAGS) -o
$@ ${MAKEFILE_PATH}kp_kernel_ldms.cpp \ $(LIBS)

kp_sampler.so: ${MAKEFILE_PATH}kp_sampler.cpp ${MAKEFILE_PATH}kp_kernel_info.h $(CXX)
$(SHARED_CXXFLAGS) $(CXXFLAGS) $(LDFLAGS) -o $@ ${MAKEFILE_PATH}kp_sampler.cpp \
$(LIBS)

clean:

rm *.so
```

To use the Kokkos connector and sampler with an application, sampling 1\% of the kokkos kernel execution times, include the following variables into your application execution script:

```
export KOKKOS_LDMS_HOST="localhost"

export KOKKOS_LDMS_PORT="411"

export KOKKOS_PROFILE_LIBRARY="<install
directory>/kokkosConnector/kp_sampler.so;<install
directory>/kokkosConnector/kp_kernel_ldms.so"

export KOKKOS_SAMPLER_RATE=101

export KOKKOS_LDMS_VERBOSE=0

export KOKKOS_LDMS_XPRT="sock"
```

Note that the KOKKOS_SAMPLER_RATE variable can be changed based on the wanted sampling rate. The current sampling rate, seen above, is set to approximately 1%. It is best to set this variable to a prime number to avoid a patterned sampling of the same kernels.

Building SOS/DSOS on the Monitoring cluster:

DSOS was used with the additional artifacts and can be obtained at the URL in (2),

SOS/DSOS database is built with the configure options shown here:

\begin{verbatim}

```
../configure CFLAGS= --prefix=/opt/ovis --libdir=/opt/ovis/lib64 --libexecdir=/opt/ovis/lib64
```

\end{verbatim}

Building LDMS on the Monitoring cluster:

LDMS is built with the configure options shown here:

\begin{verbatim}

```
../configure CFLAGS= --prefix=/opt/ovis --libdir=/opt/ovis/lib64 --libexec=/opt/ovis/lib64 --enable-ldms-python --with-sos=/opt/ovis
```

\end{verbatim}

Configuration of LDMS on the HPC system:

The LDMS sampler daemons on compute and login nodes are configured as below where “sampler.conf” will be further customized to the actual naming of system components and system data desired to be collected:

```
/opt/ovis/sbin/ldmsd -x sock:411 -c sampler.conf -v ERROR -l /var/log/ldmsd.log
```

Where the contents of the configuration file “sampler.conf” is:

\begin{verbatim}

```
env COMPONENT_ID=$(<calculate component ID here (e.g., 42)>)

env NID=$(<create node name here (e.g., foo42)>)

env SAMPLE_INTERVAL=1000000

env SAMPLE_OFFSET=0

env SLURM_JOB_SET=${NID}/slurm

load name=slurm_sampler

config name=slurm_sampler component_id=${COMPONENT_ID} producer=${NID} instance=${SLURM_JOB_SET} job_count=1

start name=slurm_sampler interval=${SAMPLE_INTERVAL} offset=0
```

```
load name=meminfo

config name=meminfo producer=${NID} instance=${NID}/meminfo component_id=${COMPONENT_ID}
job_set=${SLURM_JOB_SET} uid=0 gid=0 perm=0777

start name=meminfo interval=${SAMPLE_INTERVAL} offset=${SAMPLE_OFFSET}
```

\end{verbatim}

The LDMS first level aggregator daemons are run in the same way as the sampler daemon but with the configuration file, in this case, describing what sampler LDMS daemons to connect to (prdcr_add) and what Streams tags to subscribe to (prdcr_subscribe) as shown below (note that n in this case represents a node number left padded with zeros to 5 decimals. Name is an arbitrary string identifier and host is an actual host name or ip address):

```
/opt/ovis/sbin/ldmsd -x sock:412 -c aggregator01.conf -v ERROR -l /var/log/ldmsd_aggregator01.log
```

Where the contents of the configuration file “aggregator01.conf” is:

\begin{verbatim}

```
prdcr_add name=nid00001 type=active interval=30000000 xprt=sock host=node00001 port=411

.

.

.

prdcr_add name=nid<n> type=active interval=30000000 xprt=sock host=node<n> port=411


prdcr_subscribe stream=kokkos-perf-data regex=.*

prdcr_start_regex regex=.*


updtr_add name=all interval=1000000 offset=250000 auto_interval=true

updtr_prdcr_add name=all regex=^nid.*

updtr_start name=all
```

\end{verbatim}

Level 2 and greater aggregators are run in the same fashion with similar configuration files but with producers being the next lower-level aggregators and offsets being increased.

Configuring LDMS on the Monitoring cluster

In our case this is the second level of aggregator daemons. As this is the terminus, these daemons are additionally configured to route application messages to sosdb storage:

```
\begin{verbatim}
<producer configs as above>
load name=kokkos_appmon_store
config name=kokkos_appmon_store path=<store path> stream=kokkos-perf-data
load name=store_sos
config name=store_sos path=<store path>/LDMS_SOS
strgp_add name=meminfo-store_sos plugin=store_sos container=metric_data schema=meminfo
strgp_start name=meminfo-store_sos
\end{verbatim}
```

Building, Configuring, and Deploying Analysis and Visualization (HWS) on the Monitoring Cluster:

To deploy the HPC Web Services infrastructure sosdb-ui, sosdb-grafana, and numsos software codes are built according to the instructions in their respective github repositories. These repositories are installed into the same location; by default this is /var/www/ovis_web_svcs.

dsosds (dsos data source) software code is installed into the Grafana plugin directory, which is by default /var/lib/grafana/plugins.

The software suites sosdb-grafana, numsos, and dsosds are included in the artifacts.

An apache web server is installed on the chosen cluster to handle communication between the Grafana server and the sosdb-grafana backend. The dsosds plugin is loaded by grafana, once connection to Grafana is established via its web API, and configured to make requests to the apache host. The grafana server is configured using /etc/grafana/grafana.ini to point to this apache web server.

Python analysis modules are placed in a directory pointed to in the apache settings file /etc/httpd/conf/httpd.conf. The python modules used in this work are included in the artifacts.

The JSON object of the Grafana dashboard used is also included in the artifacts. The python modules used in this work are included in the artifacts under numsos and the AnalysisViz artifacts directory.