
Performance Analysis

Release 3.0.0

Brookhaven National Laboratory

Jul 24, 2020

INTRODUCTION

1	Overview	3
2	On-node AD Module	5
2.1	Parser	5
2.2	Pre-processing	5
2.3	Update local parameters	5
2.4	Anomaly Detection	5
2.5	Stream local viz data	6
2.6	Post-processing	6
3	Parameter Server	7
3.1	Simple Parameter Server	7
3.2	Scalable Parameter Server	8
4	Provenance Database	9
4.1	Function event schema	9
4.2	Metadata schema	11
5	Installation	13
5.1	Ubuntu 16.04	13
5.2	Summit	13
5.3	Cori	14
6	Full API Listing	15
6.1	API	15
7	Indices and tables	87
	Index	89

- Performance analysis C/C++ library
- This library is a part of [Chimbuko](#), a workflow-level scalable performance trace analysis tool.
- Funded by the [Exascale Computing Project \(ECP\)](#), U.S. Department of Energy

Related Github repositories

- [Chimbuko](#)
- [Performance Analysis](#)
- [Visualization](#)

OVERVIEW

The anomaly detection (ADM) module consists of three components: **on-node anomaly detection (AD)**, **parameter server (PS)** and **provenance database (ProvDB)**.

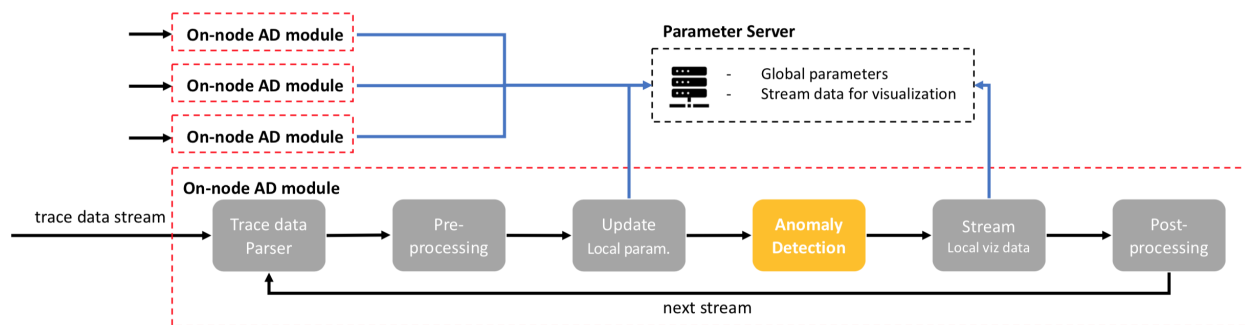


Fig. 1: Anomaly detection (AD) module: on-node AD module and paramter server (PS).

As described by the diagram above, an instrumented application communicates trace information to an instance of the AD, whose role it is to decide whether a function execution was anomalous. The decision is based upon globally aggregated function statistics that are collected on the PS and kept in sync with the AD instances. The PS also fulfils the role of collecting global statistics (number of anomalies, various counters) to forward to the external visualization module.

Detailed information about each anomaly is collected by the AD instances and forwarded to the ProvDB, which can be queried both online and offline to obtain more information.

ON-NODE AD MODULE

The on-node anomaly detection (AD) module (per applications' process) takes streamed trace data. Each AD parses the streamed trace data and maintains a function call stack along with any communication events (if available). Then, it determines anomalous function calls that have extraordinary behaviors. If there are any anomalies within the current trace data, the AD module stores them in files or DB. This is where significant data reduction occurs because we only save the anomalies and a few nearby normal function calls of the anomalies.

2.1 Parser

Currently, the trace data is streamed via [ADIOS2](#). We provide class [ADParser](#) to connect to an ADIOS2 writer side and fetch necessary data for the performance analysis.

2.2 Pre-processing

In the pre-processing step, the **on-node AD module** maintains a call stack tree in application, rank and thread levels (See class [ADEvent](#)). While it is building and maintaining the call stack tree, it computes inclusive and exclusive running time for each function, and mapping communication events to a function event.

2.3 Update local parameters

Using the pre-processed data, it first computes local parameters (depends on anomaly detection algorithm). Then, the local parameters are updated via the Parameter Server to have robust and consistent anomaly detection capabilities over the distributed **on-node AD modules**. (See [ADOutlier](#)).

2.4 Anomaly Detection

With updated anomaly detection parameters, it determines anomaly functions calls. (See [ADOutlier](#))

2.4.1 Statistical anomaly analysis

An anomaly function call is a function call that has a longer (or shorter) execution time than a upper (or a lower) threshold.

$$\begin{aligned} threshold_{upper} &= \mu_i + \alpha * \sigma_i \\ threshold_{lower} &= \mu_i - \alpha * \sigma_i \end{aligned}$$

where μ_i and σ_i are mean and standard deviation of execution time of a function i , respectively, and α is a control parameter (the lesser value, the more anomalies or the more false positives).

(See [ADOutlier](#) and [RunStats](#)).

2.4.2 Advanced anomaly analysis

TBD

2.5 Stream local viz data

Once anomalies are identified, statistics related those anomalies (e.g. mean and standard deviation of the number of anomalies per rank) is sent to the Parameter Server. Then the Parameter Server will stream the aggregated statistics to the Visualization Server so that users can evaluate the overall performance of the running applications in real time. (See [ADOutlier](#)).

2.6 Post-processing

Currently, in the post-processing step, the evaluated function calls are trimmed out from the call stack tree (See [ADE-vent](#)) and the trimmed function calls are sent to the visualization server or stored in the database according to users' configuration (See [ADio](#))

PARAMETER SERVER

The parameter server (PS) provides two services:

- Maintain global parameters to provide consistent and robust anomaly detection power over on-node AD modules
- Keep a global view of workflow-level performance trace analysis results and stream to visualization server

3.1 Simple Parameter Server

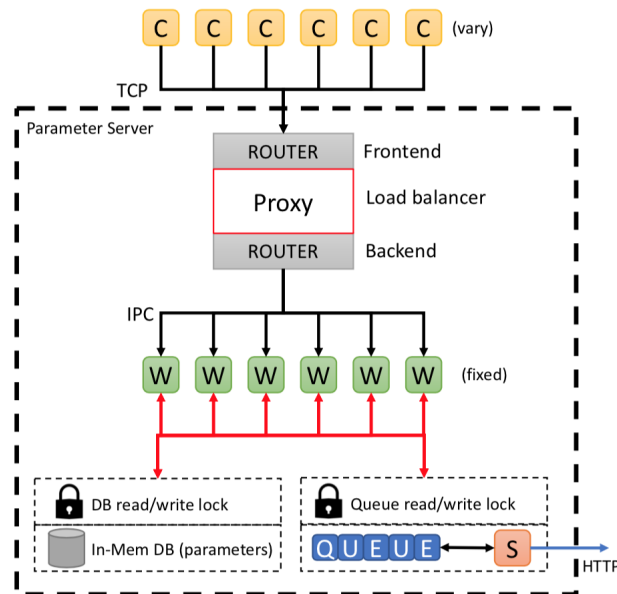


Fig. 1: Simple parameter server architecture

(C)lients (i.e. on-node AD modules) send requests with thier local parameters to be updated and to get global parameters. The requests goes to the **Frontend** router and distributed thread (W)orkers via the **Backend** router in round-robin fashion. For the task of updating parameters, workers first acquire a global lock, and update the **in-mem DB**, and return the latest parameters at the momemnt. Similrary, for the task of streaming global anomaly statistics, it will stored in a queue and the (S)treaming thread, which is dedicated to stream the anomaly statistics to a visualization server periodically.

- For network layer, see [ZMQNet](#)
- For in-Mem DB, see [SstdParam](#)

This simple parameter server becomes a bottleneck as the number of requests (or clients) are increasing. In the following subsection, we will describe the scalable parameter server.

3.2 Scalable Parameter Server

TBD

PROVENANCE DATABASE

The role of the provenance database is to store detailed information about anomalous events. For comparison, samples of normal executions are also stored. Additionally, a wealth of metadata regarding the characteristics of the devices upon which the application is running are stored within.

The database is implemented using [Sonata](#) which implements a remote database built on top of [UnQLite](#), a serverless JSON document store database. Sonata is capable of furnishing many clients and allows for arbitrarily complex document retrieval via [jx9 queries](#).

The database is organized into three *collections*:

- **anomalies** : the anomalous function executions
- **normalexecs** : the samples of normal function executions
- **metadata** : the metadata describing the machine/devices

Below we describe the JSON schema for the **anomalies** and **normalexecs** collections.

4.1 Function event schema

This section describes the JSON schema for the **anomalies** and **normalexecs** collections. The fields of the JSON object are **bolded**, and a brief description follows the colon (:).

```
{
  "__id": Record index assigned by Sonata,
  "call_stack": Function call stack,
  [
    {
      "entry_time": timestamp of function entry ,
      "fid": Global function index (can be used as a key instead of function name),
      "func": function name
    },
    ....
  ],
  "counter_events": [ An array of counter data received on the specific process thread during function execution
    {
      "counter_idx": An index used internally to index counters,
      "counter_name": A string describing the counter,
```

```
        "counter_value": The value of the counter (integer),
        "pid": process index,
        "rid": process rank,
        "tid": process thread,
        "ts": timestamp
    },
    ...
],
"entry": Timestamp of function entry,
"exit": Timestamp of function exit,
"event_id": A unique string of format "<PROCESS:RANK:INDEX>" associated with the event,
"fid": Global function index (can be used as a key instead of function name),
"func": function name,
"func_stats": Statistics of function execution time
{
    "accumulate": not used at present,
    "count": number of times function encountered (global),
    "kurtosis": kurtosis of distribution,
    "maximum": maximum of distribution,
    "mean": mean of distribution,
    "minimum": minimum of distribution,
    "skewness": skewness of distribution,
    "stddev": standard deviation of distribution
},
"is_gpu_event": true or false depending on whether function executed on a GPU
"gpu_location": if a GPU event, a JSON description of the context (see below), otherwise null,
"gpu_parent": if a GPU event, a JSON description of the parent CPU function (see below), otherwise null,
"pid": process index,
"rid": process rank,
"tid": thread index
"runtime_exclusive": Function runtime exclusive of children,
"runtime_total": Function total runtime,
}
```

The schema for the **gpu_location** field is as follows:

```
{
    "context": GPU device context (NVidia terminology),
    "device": GPU device index,
    "stream": GPU device stream (NVidia terminology),
    "thread": virtual thread index assigned to this context/device/stream by Tau
}
```

and for the **gpu_parent** field:

```
{
  "event_id": The event index string (format "<PROCESS:RANK:INDEX>") describing the parent function execution,
  "tid": Thread index for CPU parent function,
  "call_stack": Parent function call stack,
  [
    {
      "entry_time": timestamp of function entry ,
      "fid": Global function index (can be used as a key instead of function name),
      "func": function name
    },
    ....
  ]
}
```

Note that Tau considers a GPU device/context/stream much in the same way as a CPU thread, and assigns it a unique index. This index is the "thread index" for GPU events.

4.2 Metadata schema

Metadata are stored in the metadata collection in the following JSON schema:

```
{
  "descr": String description (key) of metadata entry
  "rid": Process rank from which metadata originated,
  "tid": Process thread associated with metadata,
  "value": Value of the metadata entry,
  "__id": Record index assigned by Sonata*
}
```

Note that the **tid** (thread index) for metadata is usually 0, apart from for metadata associated with a GPU context/device/stream, for which the index is the virtual thread index assigned by Tau to the context/device/stream.

INSTALLATION

For Ubuntu 16.04 system, we provide pre-built docker images users can quickly start with their own TAU instrumented applications (See [Chimbuko docker](#)).

First, download (or clone) **Chimbuko** AD module.

```
git clone https://github.com/CODARcode/PerformanceAnalysis.git
```

5.1 Ubuntu 16.04

The AD module requires to have [ADIOS2](#), [ZeroMQ](#), and [CURL](#). To install [ADIOS2](#) (MPI version), please check its website. For [ZeroMQ](#) and [CURL](#),

```
apt-get install libzmq3-dev curl libcurl4-openssl-dev
```

Optionally, to build test cases, users need to install gtest.

```
apt-get install libgtest-dev
cd /usr/src/gtest
cmake CMakeLists.txt
make
cp *.a /usr/lib
```

Finally, to build the AD module

```
cd /path/to/ad/module/dir
make
./run_test.sh # to run test cases
```

Note that users need to modify the Makefile for the [ADIOS2](#) path.

5.2 Summit

We provide an `installation` script for [ADIOS2](#), if the latest version is not available on Summit.

To load required modules and build the AD module on Summit,

```
cd /path/to/ad/module/dir
source env.summit.sh
make -f Makefile.summit
```

Note that users need to modify `Makefile.summit` for the [ADIOS2](#) path.

5.3 Cori

TBD

FULL API LISTING

6.1 API

6.1.1 AD

The “Anomaly Detection” (AD) component of Chimbuko is deployed alongside an instance of the target application (e.g. for each MPI task) and analyzes the raw trace output provided by Tau. Using globally-aggregated statistics a local decision is made as to whether a particular function execution is anomalous and the anomaly information is forwarded to the higher level components of the tool.

chimbuko

The main interface for the AD module.

namespace chimbuko

class Chimbuko

#include <chimbuko.hpp> The main interface for the AD module.

Public Functions

Chimbuko ()

~Chimbuko ()

Chimbuko (const *ChimbukoParams* ¶ms)

Construct and initialize the AD with the parameters provided.

void initialize (const *ChimbukoParams* ¶ms)

Initialize the AD with the parameters provided (must be performed prior to running)

void finalize ()

Free memory associated with AD components (called automatically by destructor)

bool use_ps () const

Whether the parameter server is in use.

void show_status (bool *verbose* = false) const

Request that the event manager print its status.

bool **get_status** () **const**

Whether the AD is connected through ADIOS2 to the trace input.

int **get_step** () **const**

Get the current IO step.

void **run** (unsigned long long &*n_func_events*, unsigned long long &*n_comm_events*, unsigned long long &*n_counter_events*, unsigned long &*n_outliers*, unsigned long &*frames*)
Run the main *Chimbuko* analysis loop.

Parameters

- [out] *n_func_events*: number of function events recorded
- [out] *n_comm_events*: number of comm events recorded
- [out] *n_counter_events*: number of counter events recorded
- [out] *n_outlier*: number of anomalous events recorded
- [out] *frames*: number of adios2 input steps

Private Functions

void **init_io** ()

void **init_parser** ()

void **init_event** ()

void **init_net_client** ()

void **init_outlier** ()

void **init_counter** ()

void **init_metadata_parser** ()

bool **parseInputStep** (int &*step*, unsigned long long &*n_func_events*, unsigned long long &*n_comm_events*, unsigned long long &*n_counter_event*)
Signal the parser to parse the adios2 timestep.

Return false if unsuccessful, true otherwise

Parameters

- [out] *step*: index
- [out] *number*: of func events parsed
- [out] *number*: of comm events parsed
- [out] *number*: of counter events parsed

void **extractEvents** (int *rank*, int *step*)

Extract parsed events and insert into the event manager.

Parameters

- *rank*: The MPI rank of the process
- *step*: The adios2 stream step index

void **extractCounters** (int *rank*, int *step*)

Extract parsed counters and insert into counter manager.

Parameters

- *rank*: The MPI rank of the process
- *step*: The adios2 stream step index

Private Members

ADParser ***m_parser**
adios2 input data stream parser

ADEvent ***m_event**
func/comm event manager

ADCounter ***m_counter**
counter event manager

ADOutlierSSTD ***m_outlier**
outlier detection algorithm

ADio ***m_io**
output writer

ADNetClient ***m_net_client**
client for comms with parameter server

ADMetadataParser ***m_metadata_parser**
parser for metadata

PerfStats **m_perf**
Performance data

ChimbukoParams **m_params**
Parameters to setup the AD

bool m_is_initialized
Whether the AD has been initialized

struct ChimbukoParams
#include <chimbuko.hpp> Parameters for setting up the AD.

Public Functions

ChimbukoParams ()

void print () const

Public Members

std::string **trace_engineType**
The ADIOS2 communications mode. If “SST” it will receive trace data in real-time, if “BPfile” it will parse an existing trace dump

std::string **trace_data_dir**
Directory containing input file.

std::string **trace_inputFile**
The input file. Assuming the environment variable TAU_FILENAME is set, the binary name is BINARY_NAME and the MPI rank is WORLD_RANK, the file format is < inputFile = “\${TAU_FILENAME}-\${BINARY_NAME}-\${WORLD_RANK}.bp” < Do not include the .sst file extensions for SST mode

double **outlier_sigma**
The number of sigma (standard deviations) away from the mean runtime for an event to be considered anomalous

std::string **pserver_addr**

The address of the parameter server. < If no parameter server is in use, this string should be empty (length zero) < If using ZmqNet (default) this is a tcp address of the form “tcp://{ADDRESS}:{PORT}”

IOMode **viz_iomode**

Set to IOMode::Online to send to viz module, IOMode::Offline to dump to disk, or IOMode::Both for both

std::string **viz_datadump_outputPath**

If writing to disk, write to this directory

std::string **viz_addr**

If sending to the viz module, this is the web address (expected <http://...>)

unsigned int **viz_anom_winSize**

When anomaly data are written, a window of this size (in units of events) around the anomalous event are also sent

std::string **perf_outputpath**

Output path for AD performance monitoring data. If an empty string no output is written.

int **perf_step**

How frequently (in IO steps) the performance data is dumped

int **rank**

MPI rank of AD process

bool **verbose**

Enable verbose output. Typically one enables this only on a single node (eg verbose = (rank==0);)

bool **only_one_frame**

Force the AD to stop after a single IO frame

int **interval_msec**

Force the AD to pause for this number of ms at the end of each IO step

ADAnomalyProvenance

namespace chimbuko

class ADAnomalyProvenance

#include <ADAnomalyProvenance.hpp> A class that gathers provenance data associated with a detected anomaly.

Public Functions

ADAnomalyProvenance(const *ExecData_t* &call, const *ADEvent* &event_man, const *ParamInterface* &func_stats, const *ADCounter* &counters, const *ADMetadataParser* &metadata)

nlohmann::json **get_json**() const

Serialize anomaly data into JSON construct.

Private Members

ExecData_t **m_call**

The anomalous event

std::vector<nlohmann::json> **m_callstack**

Call stack from function back to root. Each entry is the function index and name

nlohmann::json **m_func_stats**

JSON object containing run statistics of the anomalous function

std::vector<nlohmann::json> **m_counters**

A list of counter events that occurred during the execution of the anomalous function

bool **m_is_gpu_event**

Is this an anomaly that occurred on a GPU?

nlohmann::json **m_gpu_location**

If it was a GPU event, which device/context/stream did it occur on

nlohmann::json **m_gpu_event_parent_info**

If a GPU event, info related to CPU event spawned it (name, thread, callstack)

ADCounter

namespace chimbuko

Typedefs

typedef std::list<*CounterData_t*> **CounterDataList_t**

typedef *CounterDataList_t*::iterator **CounterDataListIterator_t**

typedef std::map<unsigned long, std::list<*CounterDataListIterator_t*>> **CounterTimeStamps_t**

typedef std::map<unsigned long, std::list<*CounterDataListIterator_t*>> **CountersByIndex_t**

Functions

DEF_MAP3UL (CounterDataListMap, *CounterDataList_t*)

map of process, rank, thread -> CounterDataList_t

DEF_MAP3UL (CounterTimeStampMap, *CounterTimeStamps_t*)

map of process, rank, thread -> CounterTimeStamps_t

class **ADCounter**

#include <ADCounter.hpp> A class that stores counter events.

Public Functions

ADCounter ()

~ADCounter ()

void **linkCounterMap** (**const** std::unordered_map<int, std::string> **m*)
pass in the pointer to the mapping of counter index to counter description

Parameters

- *m*: hash map to counter descriptions

void **addCounter** (**const** *Event_t* &*event*)

Insert a new counter.

Parameters

- *event*: *Event_t* wrapper around the counter data

CounterDataListMap_p_t **const** ***getCounters** () **const**

Return all counters collected in the timestep.

CounterDataListMap_p_t ***flushCounters** ()

Return all counters and clear internal state.

Return A pointer to a list of counters (should be deleted externally)

std::list<*CounterDataListIterator_t*> **getCountersInWindow** (**const** unsigned long *pid*,
const unsigned long *rid*,
const unsigned long *tid*,
const unsigned long *t_start*,
const unsigned long *t_end*)
const

Get counters for a particular process/rank/thread that were recorded in the window (*t_start*, *t_end*) [inclusive].

const *CountersByIndex_t* &**getCountersByIndex** () **const**

Get the map of counters by index.

Private Members

CounterDataListMap_p_t ***m_counters**

process/rank/thread -> List of counters

const std::unordered_map<int, std::string> ***m_counterMap**

counter index -> counter name map

CounterTimeStampMap_p_t **m_timestampCounterMap**

process/rank/thread -> *Ordered* map of timestamp to counter list iterator (flushed with flushCounters)

CountersByIndex_t **m_countersByIdx**

Counter index -> all instances of this counter in the timestep (flushed with flushCounters)

ADDefine

Details.

Defines

IDX_P	index of program id
IDX_R	index of rank id
IDX_T	index of thread id
IDX_E	index of event (entry/exit/send/recv) id
FUNC_EVENT_DIM	dimension of a function (timer) event vector
FUNC_IDX_F	index of function (timer) id
FUNC_IDX_TS	index of timestamp in function (timer) event
COMM_EVENT_DIM	dimension of a communication event vector
COMM_IDX_TAG	index of communication tag
COMM_IDX_PARTNER	index of communication partner
COMM_IDX_BYTES	index of communication size (in bytes)
COMM_IDX_TS	index of communication timestamp
COUNTER_EVENT_DIM	dimension of a counter event vector
COUNTER_IDX_ID	index of counter idx
COUNTER_IDX_VALUE	index of counter value
COUNTER_IDX_TS	index of counter timestamp
MAX_RUNTIME	maximum execution time of a function (or a timer)
IO_VERSION	IO version number (deprecated)

DEF_MAP3UL (NAME, BASE)

Macro to generate a 3-level map of unsigned long to objects of type BASE. The naming convention for the map levels are \${NAME}_p_t, \${NAME}_r_t, \${NAME}_t_t.

namespace chimbuko

Enums**enum ParserError**

Error kinds of the *ADParser* class

Values:

OK = 0

OK (no error)

NoFuncData = 1

Failed to fetch function data

NoCommData = 2

Failed to fetch communication data

NoCountData = 3

Failed to fetch counter data

enum EventError

Error kinds of the *ADEvent* class.

Values:

OK = 0

OK (no error)

UnknownEvent = 1

unknown event error

UnknownFunc = 2

unknown function (timer) error

CallStackViolation = 3

call stack violoation error

EmptyCallStack = 4

empty call stack error (i.e. exit before entry)

enum IOError

Error kinds of the *ADio* class.

Values:

OK = 0

OK (no error)

OutIndexRange = 1

Out of index range error

enum IOMode

I/O mode of the *ADio* class.

Values:

Off = 0

no I/O

```

Offline = 1
    offline mode, dump to files

Online = 2
    online mode, stream data

Both = 3
    both, dump to files and stream it

enum IOOpenMode
    I/O open mode of the ADio class.

    Values:

    Read = 0
        Read

    Write = 1
        Write

enum EventDataType
    event type in performance trace data

    Values:

    Unknown = 0
        unknown

    FUNC = 1
        function (timer)

    COMM = 2
        communication

    COUNT = 3
        counters

enum ListEnd
    Which end of a list/deque.

    Values:

    Back = 0

    Front = 1

```

ADEvent

```
namespace chimbuko
```

Typedefs

```

typedef std::stack<CommData_t> CommStack_t
    a stack of CommData_t

typedef std::stack<CounterData_t> CounterStack_t
    a stack of CounterData_t

typedef std::list<ExecData_t> CallList_t
    list of function calls (ExecData_t) in entry time order

```

```
typedef CallList_t::iterator CallListIterator_t
    iterator of CallList_t

typedef std::stack<CallListIterator_t> CallStack_t
    function call stack

typedef std::unordered_map<unsigned long, std::vector<CallListIterator_t>> ExecDataMap_t
    hash map of a collection of ExecData_t per function
    key is function id and value is a vector of CallListIterator_t (i.e. ExecData_t)
```

Functions

```
DEF_MAP3UL (CommStackMap, CommStack_t)
    map of process, rank, thread -> Commstack_t

DEF_MAP3UL (CounterStackMap, CounterStack_t)
    map of process, rank, thread -> Counterstack_t

DEF_MAP3UL (CallListMap, CallList_t)
    map of process, rank, thread -> CallList_t

DEF_MAP3UL (CallStackMap, CallStack_t)
    map of process, rank, thread -> CallListIterator_t
```

class ADEvent

#include <ADEvent.hpp> Event manager whose role is to correlate function entry and exit events and associate other counters with the function call.

When a function call with ENTRY signature is inserted, the event is placed on the call stack for that thread. Events associated with MPI comms and counters are also placed on their respective stacks. When a function call with EXIT signature on the same thread is inserted, a complete call is generated and placed in the call list, and all comm and counter events on their stacks are associated with that call.

Public Functions

ADEvent (bool *verbose* = false)
Construct a new *ADEvent* object.

Parameters

- *verbose*: true to print out detailed information (useful for debug)

~ADEvent ()
Destroy the *ADEvent* object.

void **linkEventType** (const std::unordered_map<int, std::string> **m*)
copy a pointer that is externally defined event type object

Parameters

- *m*: event type object (hash map)

void **linkFuncMap** (const std::unordered_map<int, std::string> **m*)
copy a pointer that is externally defined function map object

Parameters

- *m*: function map object

void **linkCounterMap** (**const** std::unordered_map<int, std::string> **m*)
 copy a pointer that is externally defined function map object

Parameters

- *m*: counter map object

const std::unordered_map<int, std::string> ***getFuncMap** () **const**
 Get the Func Map object.

Return const std::unordered_map<int, std::string>* pointer to function map object

const std::unordered_map<int, std::string> ***getEventType** () **const**
 Get the Event Type object.

Return const std::unordered_map<int, std::string>* pointer to event type object

const std::unordered_map<int, std::string> ***getCounterMap** () **const**
 Get the Counter name object.

Return const std::unordered_map<int, std::string>* pointer to counter name object

const *ExecDataMap_t* ***getExecDataMap** () **const**
 Get the Exec Data Map object.

Return const ExecDataMap_t* pointer to ExecDataMap_t object

const CallListMap_p_t ***getCallListMap** () **const**
 Get the Call List Map object.

Return const CallListMap_p_t* pointer to CallListMap_p_t object

CallListMap_p_t &**getCallListMap** ()
 Get the Call List Map object.

Return CallListMap_p_t& pointer to CallListMap_p_t object

CallListIterator_t **getCallData** (**const** std::string &*event_id*) **const**
 Get an iterator to an *ExecData_t* instance with given event index string.

throws a runtime error if the call is not present in the call-list

void **clear** ()
 clear

EventError **addEvent** (**const** *Event_t* &*event*)
 add an event

Return EventError event error code

Parameters

- *event*: function or communication event

EventError **addFunc** (**const** *Event_t* &*event*)
 add a function event

Return EventError event error code

Parameters

- *event*: function event

EventError **addComm** (**const** *Event_t* &*event*)
 add a communication event

Return EventError event error code

Parameters

- `event`: communication event

EventError **addCounter** (**const** *Event_t* &*event*)

add a counter event

Return `EventError` event error code

Parameters

- `event`: counter event

CallListIterator_t **addCall** (**const** *ExecData_t* &*exec*)

Add a complete function call, primarily for testing.

Return Iterator to inserted call

Parameters

- `exec`: Instance of *ExecData_t*

CallListMap_p_t ***trimCallList** ()

trim out all function calls that are completed (i.e. a pair of ENTRY and EXIT events are observed)

Return *CallListMap_p_t** trimmed function calls

void **show_status** (bool *verbose* = false) **const**

show current call stack tree status

Parameters

- `verbose`: true to see all details

const `std::unordered_map<unsigned long, CallListIterator_t>` &**getUnmatchCorrelationIDevents** ()

const

Get the map of correlation ID to event for those events that have yet to be partnered.

Private Functions

void **checkAndMatchCorrelationID** (*CallListIterator_t* *it*)

Check if the event has a correlation ID counter, if so try to match it to an outstanding unmatched event with a correlation ID.

Private Members

const `std::unordered_map<int, std::string>` ***m_funcMap**

pointer to map of function index to function name

const `std::unordered_map<int, std::string>` ***m_eventType**

pointer to map of event index to event type string

const `std::unordered_map<int, std::string>` ***m_counterMap**

pointer to map of counter index to counter name string

CommStackMap_p_t **m_commStack**

communication event stack. Once a function call has exited, all comms events are associated with that call and the stack is cleared

CounterStackMap_p_t **m_counterStack**

map of process,rank,thread to counter events. Once a function call has exited, all counter events are associated with that call and the stack is cleaned.

CallStackMap_p_t m_callStack

map of process,rank,thread to the current function call stack. As functions exit they are popped from the stack

CallListMap_p_t m_callList

map of process,rank,thread to a list of *ExecData_t* objects which contain entry/exit timestamps for function calls

In practise the call list is purged of completed events each IO step through calls to trimCallList

ExecDataMap_t m_execDataMap

map of function index to an array of complete calls to this function during this IO step

In practise this map is cleared every IO step by calls to trimCallList

std::unordered_map<std::string, *CallListIterator_t*> m_callIDMap

map of call event index string to the event

Completed calls are removed from this list every IO step by calls to trimCallList

std::unordered_map<unsigned long, *CallListIterator_t*> m_unmatchedCorrelationID

Events with unmatched correlation IDs.

Events that correspond to GPU kernel launches and executions are given correlation IDs as counters that allow us to match the CPU thread that launched them to the GPU kernel event

bool m_verbose

verbose

ADglobalFunctionIndexMap

namespace chimbuko

class ADglobalFunctionIndexMap

#include <ADglobalFunctionIndexMap.hpp> A class that maintains a mapping of a local function index to a global function index that is specified by the parameter server.

If the parameter server is not connected it will simply return the local index

Public Functions

ADglobalFunctionIndexMap (*ADNetClient* *net_client = nullptr)

bool connectedToPS () const

Check if the pserver is connected.

void linkNetClient (*ADNetClient* *net_client)

Link the net client.

unsigned long lookup (**const** unsigned long local_idx, **const** std::string &func_name)

Lookup the global index corresponding to the input local index.

Function names must be unique

unsigned long lookup (**const** unsigned long local_idx) **const**

Lookup the global index corresponding to the input local index (const version; throws if not already present)

ADNetClient ***getNetClient** ()

Return a pointer to the net client.

Private Members

ADNetClient ***m_net_client**

std::unordered_map<unsigned long, unsigned long> **m_idxmap**

ADio

namespace chimbuko

class ADio

#include <ADio.hpp> A class that manages communication of JSON-formatted data to the parameter server via CURL and/or to disk.

Public Functions

ADio ()

~ADio ()

void **setRank** (int *rank*)

int **getRank** () **const**

bool **open_curl** (std::string *url*)

void **close_curl** ()

void **setOutputPath** (std::string *path*)

std::string **getOutputPath** () **const**

void **setDispatcher** (std::string *name* = "ioDispatcher", size_t *thread_cnt* = 1)

void **setWinSize** (unsigned int *winSize*)

unsigned int **getWinSize** () **const**

CURL ***getCURL** ()

std::string **getURL** ()

size_t **getNumIOJobs** () **const**

IOError **write** (CallListMap_p_t **m*, long long *step*)

Write anomalous events discovered during timestep.

Parameters

- *m*: Organized list of anomalous events
- *step*: adios2 io step

IOError **writeCounters** (CounterDataListMap_p_t **counterList*, long long *step*)

Write counter data.

Parameters

- `counterList`: List of counter events
- `adios2`: io step

IOError **writeMetadata** (**const** std::vector<*MetaData_t*> &*newMetadata*, long long *step*)

Write metadata accumulated during this IO step.

Parameters

- *newMetadata*: Vector of *MetaData_t* instances containing metadata accumulated during this IO step
- `adios2`: io step

void **setDestructorThreadWaitTime** (**const** int *secs*)

Set the amount of time between completion of thread dispatcher tasks and destruction of the dispatcher in the class destructor.

Parameters

- *secs*: The time in seconds

Private Functions

void **_open** (std::fstream &*f*, std::string *filename*, *IOOpenMode* *mode*)

Private Members

unsigned int **m_execWindow**

std::string **m_outputPath**

DispatchQueue ***m_dispatcher**

CURL ***m_curl**

std::string **m_url**

int **m_rank**

int **destructor_thread_waittime**

ADLocalCounterStatistics

namespace **chimbuko**

class ADLocalCounterStatistics

#include <ADLocalCounterStatistics.hpp> A class that gathers local counter statistics and communicates them to the parameter server.

Parameters

- *step*: The current io step
- *which_counters*: The set of counters we are interested in (not all might appear in any given run). If nullptr all counters are accepted.
- *perf*: A pointer to a *PerfStats* instance for performance data monitoring

Public Functions

ADLocalCounterStatistics(**const** int *step*, **const** std::unordered_set<std::string> **which_counters*, *PerfStats* **perf* = nullptr)

void **gatherStatistics** (**const** *CountersByIndex_t* &*cntrs_by_idx*)
Add counters to internal statistics.

std::pair<size_t, size_t> **updateGlobalStatistics** (*ADNetClient* &*net_client*) **const**
update (send) counter statistics gathered during this io step to the connected parameter server

The message string is the output of *get_json_state()* in string format

Return std::pair<size_t, size_t> [sent, recv] message size

Parameters

- *net_client*: The network client object

void **linkPerf** (*PerfStats* **perf*)
Attach a *PerfStats* object into which performance metrics are accumulated.

const std::unordered_map<std::string, *RunStats*> &**getStats** () **const**
Get the map of counter name to statistics.

nlohmann::json **get_json_state** () **const**
Get the JSON object that is sent to the parameter server.

The string form of this object is sent to the pserver using *updateGlobalStatistics*

void **setStats** (**const** std::string &*counter*, **const** *RunStats* &*to*)
Set the statistics for a particular counter (must be in the list of counters being collected). Primarily used for testing.

Protected Attributes

int **m_step**
io step

const std::unordered_set<std::string> **m_which_counter**

std::unordered_map<std::string, *RunStats*> **m_stats**
The set of counters whose statistics we are accumulating map of counter to statistics

PerfStats **m_perf**
Store performance data

Protected Static Functions

static std::pair<size_t, size_t> **updateGlobalStatistics** (*ADNetClient* &*net_client*,
const std::string &*l_stats*, int
step)

update (send) counter statistics gathered during this io step to the connected parameter server

Return std::pair<size_t, size_t> [sent, recv] message size

Parameters

- *net_client*: The network client object
- *l_stats*: local statistics
- *step*: step (or frame) number

ADLocalFuncStatistics

```
namespace chimbuko
```

```
class ADLocalFuncStatistics
```

#include <ADLocalFuncStatistics.hpp> A class that gathers local function statistics and communicates them to the parameter server.

Public Functions

```
ADLocalFuncStatistics (const int step, PerfStats *perf = nullptr)
```

```
void gatherStatistics (const ExecDataMap_t *exec_data)
```

Add function executions to internal statistics.

```
void gatherAnomalies (const Anomalies &anom)
```

Add anomalies to internal statistics.

```
std::pair<size_t, size_t> updateGlobalStatistics (ADNetClient &net_client) const
```

update (send) function statistics (#anomalies, incl/excl run times) gathered during this io step to the connected parameter server

The message communicated is the string dump of the output of *get_json_state()*

Return std::pair<size_t, size_t> [sent, recv] message size

Parameters

- *net_client*: The network client object

```
nlohmann::json get_json_state (const int rank) const
```

Get the current state as a JSON object.

The string dump of this object is the serialized form sent to the parameter server

Parameters

- *rank*: The rank of this AD instance

```
void linkPerf (PerfStats *perf)
```

Attach a *RunMetric* object into which performance metrics are accumulated.

Protected Attributes

```
int m_step
```

io step

```
unsigned long m_min_ts
```

```
unsigned long m_max_ts
```

lowest timestamp

```
std::unordered_map<unsigned long, std::string> m_func
```

highest timestamp map of function index to function name

```
std::unordered_map<unsigned long, RunStats> m_inclusive
```

map of function index to function call time including child calls

```
std::unordered_map<unsigned long, RunStats> m_exclusive
```

map of function index to function call time excluding child calls

`std::unordered_map<unsigned long, size_t> m_anomaly_count`
map of function index to number of anomalies

`size_t m_n_anomalies`
Number of anomalies in total

PerfStats *`m_perf`
Store performance data

Protected Static Functions

static `std::pair<size_t, size_t> updateGlobalStatistics` (*ADNetClient* &*net_client*,
const `std::string` &*l_stats*, `int` *step*)
update (send) function statistics (#anomalies, incl/excl run times) gathered during this io step to the
connected parameter server

Return `std::pair<size_t, size_t>` [sent, recv] message size

Parameters

- *net_client*: The network client object
- *l_stats*: local statistics
- *step*: step (or frame) number

ADMetadataParser

`namespace chimbuko`

class `ADMetadataParser`

#include <ADMetadataParser.hpp> A class that parses and maintains useful metadata.

Public Functions

`void addData` (**const** `std::vector<Metadata_t>` &*new_metadata*)
Add new metadata collected during this timeframe.

const `std::unordered_map<unsigned long, GPUvirtualThreadInfo>` &**getGPUthreadMap** ()
const

`bool isGPUthread` (**const** `unsigned long` *thr*) **const**

const *GPUvirtualThreadInfo* &**getGPUthreadInfo** (**const** `unsigned long` *thread*) **const**
Return the thread info struct for this thread. Throws an error if an invalid thread.

const `std::unordered_map<int, std::unordered_map<std::string, std::string>>` &**getGPUproperties** ()
const
Get the map of CUDA device index to a key/value pair of GPU properties.

Private Functions

void **parseMetadata** (const *MetaData_t* &m)
Parse an individual metadata entry.

Private Members

std::unordered_map<unsigned long, *GPUvirtualThreadInfo*> **m_gpu_thread_map**
Map of tau's virtual thread index to CUDA device/context/stream

std::unordered_map<int, std::unordered_map<std::string, std::string>> **m_gpu_properties**
Properties of GPU device. Index is the CUDA device index

struct GPUvirtualThreadInfo

#include <ADMetadataParser.hpp> Structure containing the CUDA device/context/stream associated with a given virtual thread index.

Public Functions

GPUvirtualThreadInfo (unsigned long *_thread*, int *_device*, int *_context*, int *_stream* = 0)

nlohmann::json **get_json** () const
Get the data as a JSON object.

Public Members

unsigned long **thread**
The virtual thread index assigned by Tau

int **device**
The device index (assigned by the CUDA runtime)

int **context**
The device context

int **stream**
Stream index if multiple streams are in use. Defaults to 0 if only one stream

ADNetClient

namespace **chimbuko**

class ADNetClient

#include <ADNetClient.hpp> A wrapper class to facilitate communications between the AD and the parameter server.

Public Functions

ADNetClient ()

bool **use_ps** () **const**

check if the parameter server is in use

Return true if the parameter server is in use

Return false if the parameter server is not in use

void **connect_ps** (int *rank*, int *srank* = 0, std::string *sname* = "MPINET")

connect to the parameter server

Parameters

- *rank*: this process rank
- *srank*: server process rank. If using ZMQnet this is not applicable
- *sname*: server name. If using ZMQNet this is the server ip address, for MPINET it is not applicable

void **disconnect_ps** ()

disconnect from the connected parameter server

int **get_server_rank** () **const**

Return the MPI rank of the parameter server.

int **get_client_rank** () **const**

Return the MPI rank of this client.

std::string **send_and_receive** (const *Message* &*msg*)

Send a message to the parameter server and receive the response.

Return The response message in string format. This is a JSON object with 'Header' and 'Buffer' fields

Parameters

- *msg*: The message

Private Members

bool **m_use_ps**

true if the parameter server is in use

int **m_rank**

int **m_srank**

server process rank

void ***m_context**

ZeroMQ context

void ***m_socket**

ZeroMQ socket

ADOutlier

namespace chimbuko

class ADOutlier

#include <ADOutlier.hpp> abstract class for anomaly detection algorithms

Subclassed by *chimbuko::ADOutlierSSTD*

Public Functions

ADOutlier()

Construct a new *ADOutlier* object.

virtual ~ADOutlier()

Destroy the *ADOutlier* object.

bool use_ps() const

check if the parameter server is in use

Return true if the parameter server is in use

Return false if the parameter server is not in use

void linkExecDataMap(const ExecDataMap_t *m)

copy a pointer to execution data map

See *ADEvent*

Parameters

- m:

void linkNetworkClient(ADNetClient *client)

Link the interface for communicating with the parameter server.

virtual Anomalies run(int step = 0) = 0

abstract method to run the implemented anomaly detection algorithm

Return data structure containing information on captured anomalies

Parameters

- step: step (or frame) number

void linkPerf(PerfStats *perf)

If linked, performance information on the sync_param routine will be gathered.

ParamInterface **const *get_global_parameters() const**

Get the local copy of the global parameters.

Return Pointer to a *ParamInterface* object

Protected Functions

virtual unsigned long **compute_outliers** (*Anomalies* &outliers, **const** unsigned long
func_id, std::vector<CallListIterator_t> &data)
= 0

abstract method to compute outliers (or anomalies)

Return unsigned long the number of outliers (or anomalies)

Parameters

- [out] outliers: data structure containing captured anomalies
- func_id: function id
- [inout] data: a list of function calls to inspect. Entries will be tagged as outliers

virtual std::pair<size_t, size_t> **sync_param** (*ParamInterface* **const** *param) = 0

abstract method to update local parameters and get global ones

Return std::pair<size_t, size_t> [sent, rcv] message size

Parameters

- [in] param: local parameters

Protected Attributes

int **m_rank**

this process rank

bool **m_use_ps**

true if the parameter server is in use

ADNetClient ***m_net_client**

interface for communicating to parameter server

std::unordered_map<std::array<unsigned long, 4>, size_t, ArrayHasher<unsigned long, 4>> **m_local_func_exec_cou**

Map(program id, rank id, thread id, func id) -> number of times encountered on this node

const *ExecDataMap_t* ***m_execDataMap**

execution data map

ParamInterface ***m_param**

global parameters (kept in sync with parameter server)

PerfStats ***m_perf**

class **ADOutlierSSTD**:**public** *chimbuko::ADOutlier*

#include <ADOutlier.hpp> statistic analysis based anomaly detection algorithm

Public Functions

ADOutlierSSTD ()

Construct a new *ADOutlierSSTD* object.

~ADOutlierSSTD ()

Destroy the *ADOutlierSSTD* object.

void **set_sigma** (double *sigma*)

Set the sigma value.

Parameters

- sigma: sigma value

Public Functions

ADParser (std::string *inputFile*, std::string *engineType* = "BPFile", int *openTimeoutSeconds* = 60)
Construct a new *ADParser* object.

Parameters

- *inputFile*: ADIOS2 BP filename
- *engineType*: BPFile or SST
- *openTimeoutSeconds*: Timeout for opening ADIOS2 stream

~ADParser ()
Destroy the *ADParser* object.

void **linkNetClient** (*ADNetClient* **net_client*)
Link the net client to the object that maintains a mapping of local function index to global index.
If this is performed, the parser will replace the local with global index in the incoming data stream

void **linkPerf** (*PerfStats* **perf*)
If linked, performance information will be gathered.

const std::unordered_map<int, std::string> ***getFuncMap** () **const**
Get the function hash map (function id > function name)
Return const std::unordered_map<int, std::string>* function hash map

const std::unordered_map<int, std::string> ***getEventType** () **const**
Get the event type hash map (event type id > event name)
Return const std::unordered_map<int, std::string>* event type hash map

const std::unordered_map<int, std::string> ***getCounterMap** () **const**
Get the counter hash map (counter id > counter description)
Return const std::unordered_map<int, std::string>* event type hash map

bool **getStatus** () **const**
Get the status of this parser.
Return true if it is connected with a writer
Return false if it is disconnected or there are no available data anymore

int **getCurrentStep** () **const**
Get the current step (or frame) number.
Return int step number

int **beginStep** (bool *verbose* = false)
start fetching next available data
Return int current step number
Parameters

- *verbose*: true to output additional information

void **endStep** ()
end current step (or frame), only effect on ADIOS2 SST engine

void **update_attributes** ()
update attributes (or meta data), with ADIOS2 BPFile engine it only fetches the available attributes one time.

ParserError **fetchFuncData** ()

fetching function (timer) data. Results stored internally and extracted using *ADParser::getFuncData*

Return ParserError error code

ParserError **fetchCommData** ()

fetching communication data. Results stored internally and extracted using *ADParser::getCommData*

Return ParserError error code

ParserError **fetchCounterData** ()

fetching counter data. Results stored internally and extracted using *ADParser::getCounterData*

Return ParserError error code

const unsigned long ***getFuncData** (size_t *idx*) **const**

get pointer to an array of a function event specified by *idx*

Return pointer to a function event array

Parameters

- *idx*: index of a function event

size_t **getNumFuncData** () **const**

Get the number of function events in the current step.

Return size_t the number of function events

const unsigned long ***getCommData** (size_t *idx*) **const**

get pointer to a communication event array specified by *idx*

Return pointer to a communication event array

Parameters

- *idx*: index of a communication event

size_t **getNumCommData** () **const**

Get the number of communication events in the current step.

Return size_t the number of communication events

const unsigned long ***getCounterData** (size_t *idx*) **const**

get pointer to a counter event array specified by *idx*

Return pointer to a counter event array

Parameters

- *idx*: index of a counter event

size_t **getNumCounterData** () **const**

Get the number of counter events in the current step.

Return size_t the number of counter events

const std::vector<*MetaData_t*> &**getNewMetaData** () **const**

Get metadata parsed for the first time during the current step.

std::vector<*Event_t*> **getEvents** (**const** int *rank*) **const**

Get all the events (func, comm and counter) occurring in the IO step ordered by their timestamp.

Parameters

- *rank*: The MPI rank of the AD process

void **addFuncData** (unsigned long **const** **d*)

For testing purposes, add the data in the array *d* to the internal `m_event_timestamps` array.

Will throw an error if the new array size exceeds the vector capacity as this would invalidate previous *Event_t* objects

Parameters

- *d*: An array of length `FUNC_EVENT_DIM`

void **addCounterData** (unsigned long **const** **d*)

For testing purposes, add the data in the array *d* to the internal `m_counter_timestamps` array.

Will throw an error if the new array size exceeds the vector capacity as this would invalidate previous *Event_t* objects

Parameters

- *d*: An array of length `COUNTER_EVENT_DIM`

void **addCommData** (unsigned long **const** **d*)

For testing purposes, add the data in the array *d* to the internal `m_comm_timestamps` array.

Will throw an error if the new array size exceeds the vector capacity as this would invalidate previous *Event_t* objects

Parameters

- *d*: An array of length `COMM_EVENT_DIM`

void **setFuncDataCapacity** (size_t *cap*)

Set the `m_event_timestamps` vector capacity in units of `FUNC_EVENT_DIM`. This will invalidate previous *Event_t* objects if it requires a realloc!

void **setCommDataCapacity** (size_t *cap*)

Set the `m_comm_timestamps` vector capacity in units of `COMM_EVENT_DIM`. This will invalidate previous *Event_t* objects if it requires a realloc!

void **setCounterDataCapacity** (size_t *cap*)

Set the `m_counter_timestamp` vector capacity in units of `COUNTER_EVENT_DIM`. This will invalidate previous *Event_t* objects if it requires a realloc!

void **setFuncMap** (**const** std::unordered_map<int, std::string> &*m*)

Set the function index->name map for testing.

void **setEventTypeMap** (**const** std::unordered_map<int, std::string> &*m*)

Set the function event index -> event type map for testing.

void **setCounterMap** (**const** std::unordered_map<int, std::string> &*m*)

Set the counter index->name map for testing.

unsigned long **getGlobalFunctionIndex** (**const** unsigned long *local_idx*) **const**

Get the global index corresponding to a given local function index. 1<->1 mapping if pserver not connected.

Private Functions

`std::pair<Event_t, bool> createAndValidateEvent (const unsigned long *data, Event-
DataType t, size_t idx, std::string id, int
rank) const`

Create an *Event_t* instance from the data at the provided pointer and run simple validation.

Private Members

`adios2::ADIOS m_ad`
adios2 handler

`adios2::IO m_io`
adios2 I/O handler

`adios2::Engine m_reader`
adios2 engine handler

`std::string m_inputFile`
adios2 BP filename

`std::string m_engineType`
adios2 engine type

`bool m_status`
parser status

`bool m_opened`
true if connected to a writer or a BP file

`bool m_attr_once`
true for BP engine

`int m_current_step`
current step

`std::unordered_set<std::string> m_metadata_seen`
Metadata descriptions that have been seen

`std::vector<MetaData_t> m_new_metadata`
New metadata that appeared on this step

`std::unordered_map<int, std::string> m_funcMap`
function hash map (function id > function name)

`std::unordered_map<int, std::string> m_eventType`
event type hash map (event type id > event name)

`std::unordered_map<int, std::string> m_counterMap`
counter hash map (counter id > counter name)

`size_t m_timer_event_count`
the number of function events in current step

`std::vector<unsigned long> m_event_timestamps`
array of all function events in the current step

`size_t m_comm_count`
the number of communication events in current step

`std::vector<unsigned long> m_comm_timestamps`
array of all communication events in the current step

`size_t m_counter_count`
the number of counter events in the current step

`std::vector<unsigned long> m_counter_timestamps`
array of all counter events in the current step

ADglobalFunctionIndexMap `m_global_func_idx_map`
Maintains mapping of local function index to global function index (if pserver connected)

PerfStats `*m_perf`
Performance monitoring

Private Static Functions

static const unsigned long **getEarliest* (**const** std::vector<**const** unsigned long ***> &arrays, **const** std::vector<int> &ts_offsets)

Return the pointer to the array whose timestamp (given by the value in the array at the provided offset) is earliest.

Some (but not all) arrays can be nullptr. If there is a tie between two entries, the array that enters first (lowest index) in the input vectors is chosen.

Parameters

- arrays: A vector of array pointers
- ts_offsets: The elements of the arrays that correspond to the timestamp

ADProvenanceDBclient

ADProvenanceDBengine

AnomalyStat

`namespace chimbuko`

Functions

`bool operator==` (**const** *AnomalyData* &a, **const** *AnomalyData* &b)

`bool operator!=` (**const** *AnomalyData* &a, **const** *AnomalyData* &b)

class *AnomalyData*

#include <AnomalyStat.hpp> A class that contains data on the number of anomalies collected during the present timestep. It contains the number of anomalies and the timestamp window in which the anomalies occurred.

Public Functions

AnomalyData ()

AnomalyData (unsigned long *app*, unsigned long *rank*, unsigned *step*, unsigned long *min_ts*, unsigned long *max_ts*, unsigned long *n_anomalies*, std::string *stat_id* = "")

AnomalyData (const std::string &*s*)

~AnomalyData ()

void **set** (unsigned long *app*, unsigned long *rank*, unsigned *step*, unsigned long *min_ts*, unsigned long *max_ts*, unsigned long *n_anomalies*, std::string *stat_id* = "")

unsigned long **get_app** () const

unsigned long **get_rank** () const

unsigned long **get_step** () const

unsigned long **get_min_ts** () const

unsigned long **get_max_ts** () const

unsigned long **get_n_anomalies** () const

std::string **get_stat_id** () const

nlohmann::json **get_json** () const

Private Members

unsigned long **m_app**

unsigned long **m_rank**

unsigned long **m_step**

unsigned long **m_min_timestamp**

unsigned long **m_max_timestamp**

unsigned long **m_n_anomalies**

std::string **m_stat_id**

Friends

bool **operator==** (const AnomalyData &*a*, const AnomalyData &*b*)

bool **operator!=** (const AnomalyData &*a*, const AnomalyData &*b*)

class AnomalyStat

#include <AnomalyStat.hpp> A class that contains statistics on the number of anomalies detected.

Public Functions

AnomalyStat (bool *do_accumulate* = false)

~AnomalyStat ()

void **add** (*AnomalyData* &*d*, bool *bStore* = true)

Add the anomaly count from the input *AnomalyData* instance to the internal statistics.

Parameters

- *d*: The *AnomalyData* instance
- *bStore*: If true the *AnomalyData* instance dumped to a JSON-formatted string will be added to the “data list”

void **add** (const std::string &*str*, bool *bStore* = true)

Add the anomaly count from the input string, a JSON-formatted dump of an *AnomalyData* instance, to the internal statistics.

Parameters

- *d*: The *AnomalyData* instance
- *bStore*: If true the string will be added to the “data list”

std::pair<*RunStats*, std::list<std::string>*> **get** ()

Get copy of the current statistics and the pointer to data list.

WARN: Once this function is called, the pointer to the current data list is returned and new (empty) data list is allocated. So, it is callee’s responsibility to free the allocated memory.

Return std::pair<*RunStats*, std::list<std::string>*>

RunStats **get_stats** ()

Return a copy of current statistics.

Note: this function does not return a reference because the internal state is constantly changing. Here we temporarily lock the state while generating the copy

std::list<std::string> ***get_data** ()

Get the pointer to the data list.

WARN: As it returns the pointer to the data list, new data can be added to the list in other threads. Also, it shouldn’t be freed by the callee.

Return std::list<std::string>*

size_t **get_n_data** () const

Private Members

std::mutex **m_mutex**

RunStats **m_stats**

std::list<std::string> ***m_data**

Statistics on the number of anomalies over all ranks A list of JSON-formatted strings that represent serializations of the incoming *AnomalyData* instances since last flush

ExecData

namespace chimbuko

Functions

bool **operator<** (const *Event_t* &lhs, const *Event_t* &rhs)
compare two events

bool **operator>** (const *Event_t* &lhs, const *Event_t* &rhs)
compare two events

class **CommData_t**
#include <ExecData.hpp> wrapper for communication event

Public Functions

CommData_t ()
Construct a new *CommData_t* object.

CommData_t (const *Event_t* &ev, std::string commType)
Construct a new *CommData_t* object.

Parameters

- ev: constant reference to a *Event_t* object
- commType: communication type (e.g. SEND/RECV)

~CommData_t ()
Destroy the *CommData_t* object.

std::string **type** () const
return communication type

unsigned long **ts** () const
return timestamp

unsigned long **src** () const
return source process id of this communication event

unsigned long **tar** () const
return target (or destination) process id of this communication event

void **set_exec_key** (std::string key)
Set the execution key id (i.e. where this communication event occurs). This is equal to the “id” string associated with a parent *ExecData_t* object.

Parameters

- key: execution id

const std::string &**get_exec_key** () const
Get the execution key id. This is equal to the “id” string associated with a parent *ExecData_t* object.

bool **is_same** (const *CommData_t* &other) const
compare two communication data

Return true if other is same

Return false if other is different

Parameters

- other:

nlohmann::json **get_json()** **const**

Get the json object of this communication data.

Private Members

std::string **m_commType**

communication type

unsigned long **m_pid**

program id

unsigned long **m_rid**

rank id

unsigned long **m_tid**

thread id

unsigned long **m_src**

source process id

unsigned long **m_tar**

target process id

unsigned long **m_bytes**

communication data size in bytes

unsigned long **m_tag**

communication tag

unsigned long **m_ts**

communication timestamp

std::string **m_execkey**

execution key (or id) where this communication event occurs

class CounterData_t

#include <ExecData.hpp> wrapper for counter event

Public Functions

CounterData_t()

Construct a new *CounterData_t* object.

CounterData_t(const Event_t &ev, const std::string &counter_name)

Construct a new *CounterData_t* object.

Parameters

- ev: constant reference to a *Event_t* object
- commType: communication type (e.g. SEND/RECV)

nlohmann::json **get_json()** **const**

Get the json object of this communication data.

unsigned long **get_pid()** **const**

return program id

unsigned long **get_rid()** **const**
return rank id

unsigned long **get_tid()** **const**
return thread id

const std::string &**get_countersname()** **const**
Return the name of the counter.

unsigned long **get_value()** **const**
Return the value of the counter.

unsigned long **get_ts()** **const**
Return the counter timestamp.

unsigned long **get_counterid()** **const**
Return the index of the counter.

void **set_exec_key**(std::string *key*)
Set the execution key id (i.e. where this counter event occurs). This is equal to the “id” string associated with a parent *ExecData_t* object.

Parameters

- *key*: execution id

const std::string &**get_exec_key()** **const**
Get the execution key id. This is equal to the “id” string associated with a parent *ExecData_t* object.

Private Members

std::string **m_countersname**
counter name

unsigned long **m_pid**
program id

unsigned long **m_rid**
rank id

unsigned long **m_tid**
thread id

unsigned long **m_cid**
counted id

unsigned long **m_value**
counter value

unsigned long **m_ts**
counter timestamp

std::string **m_execkey**
execution key (or id) where this counter event occurs

class Event_t

#include <ExecData.hpp> class to provide easy access to raw performance event vector

The data are passed in via ADIOS2 and stored internally in a compressed format in the form of an integer array, blocks of which are associated with particular events. Each block has a certain number of entries

associated with it that relate to information such as program, comm and thread index, timestamp as well as detailed event information. The mappings are set out in `ADDefine.hpp`.

This class wraps the event data blocks allowing for retrieval of event information through explicit function calls. It works for all event types: function, comm and counter

Public Functions

Event_t (**const** unsigned long **data*, *EventDataType* *t*, size_t *idx*, std::string *id* = "event_id")
Construct a new *Event_t* object.

Parameters

- *data*: pointer to raw performance event vector
- *t*: event type
- *idx*: event index
- *id*: event (string) id

~Event_t ()
Destroy the *Event_t* object.

bool **valid** () **const**
check if the raw data pointer is valid

std::string **id** () **const**
return event id

size_t **idx** () **const**
return event index, typically the index of the event in the input array for the timestep on which it was spawned

unsigned long **pid** () **const**
return program id

unsigned long **rid** () **const**
return rank id

unsigned long **tid** () **const**
return thread id

unsigned long **eid** () **const**
return event type id (FUNC/COMM only). Eg for FUNC events is is ENTRY/EXIT

unsigned long **ts** () **const**
return timestamp of this event

EventDataType **type** () **const**
return event type

std::string **strtype** () **const**
return string event type

unsigned long **fid** () **const**
return function (timer) id (FUNC event only)

unsigned long **tag** () **const**
return communication tag id (COMM event only)

unsigned long **partner** () **const**
return communication partner id (COMM event only)

unsigned long **bytes** () **const**
return communication data size (in bytes) (COMM event only)

unsigned long **counter_id** () **const**
return counter id (COUNT event only)

unsigned long **counter_value** () **const**
return the value of the counter (COUNT event only)

bool **operator==** (const *Event_t* &r) **const**
Equivalence operation.

Note the underlying array pointers can be different providing the values are identical

nlohmann::json **get_json** () **const**
Get the json object of this event object.

const unsigned long ***get_ptr** () **const**
Return the pointer to the underlying data.

int **get_data_len** () **const**
Get the length of the underlying array.

Private Members

const unsigned long ***m_data**
pointer to raw performance trace data vector

EventDataType **m_t**
event type

std::string **m_id**
event id

size_t **m_idx**
event index

Friends

bool **operator<** (const *Event_t* &lhs, const *Event_t* &rhs)
compare two events

bool **operator>** (const *Event_t* &lhs, const *Event_t* &rhs)
compare two events

class ExecData_t
#include <ExecData.hpp> A pair of function (timer) events, ENTRY and EXIT.

Public Functions

ExecData_t ()

Construct a new *ExecData_t* object.

ExecData_t (const *Event_t* &ev)

Construct a new *ExecData_t* object.

Parameters

- *ev*: constant reference to a *Event_t* object

~ExecData_t ()

Destroy the *ExecData_t* object.

std::string get_id () const

Get the id of this execution data.

std::string get_funcname () const

Get the function name of this execution data.

unsigned long get_pid () const

Get the program id of this execution data.

unsigned long get_tid () const

Get the thread id of this execution data.

unsigned long get_rid () const

Get the rank id of this execution data.

unsigned long get_fid () const

Get the function id of this execution data.

long get_entry () const

Get the entry time of this execution data.

long get_exit () const

Get the exit time of this execution data.

long get_runtime () const

Get the (inclusive) running time of this execution data.

long get_inclusive () const

Get the (inclusive) running time of this execution data.

long get_exclusive () const

Get the exclusive running ime of this execution data.

int get_label () const

Get the label of this execution data.

Return int 1 of normal and -1 os anomaly

std::string get_parent () const

Get the parent function id of this execution data.

const std::deque<*CommData_t*> &get_messages () const

Get a list of communication data occurred in this execution data.

const std::deque<*CounterData_t*> &**get_counters** () **const**

Get a list of counter events that occurred in this execution data.

unsigned long **get_n_message** () **const**

Get the number of communication events.

unsigned long **get_n_children** () **const**

Get the number of child functions.

unsigned long **get_n_counter** () **const**

Get the number of counter.

void **set_label** (int *label*)

Set the label.

Parameters

- *label*: 1 for normal, -1 for anomaly

void **set_parent** (std::string *parent*)

Set the parent function of this execution.

Parameters

- *parent*: the parent execution id

void **set_funcname** (std::string *funcname*)

Set the function name of this execution.

Parameters

- *funcname*: function name

bool **update_exit** (**const** *Event_t* &*ev*)

update exit event of this execution

Return true no errors

Return false incorrect exit event

Parameters

- *ev*: exit event

void **update_exclusive** (long *t*)

update exclusive running time

Parameters

- *t*: running time of a child function

void **inc_n_children** ()

increase the number of child function by 1

bool **add_message** (**const** *CommData_t* &*comm*, *ListEnd* *end* = *ListEnd::Back*)

add communication data to one end of the message queue

Return true no errors

Return false invalid communication event

Parameters

- *comm*: communication event occurred in this execution
- *end*: add to which end of the deque

bool **add_counter** (**const** *CounterData_t* &*count*, *ListEnd* *end* = *ListEnd::Back*)

add counter data

Return true no errors

Return false invalid communication event

Parameters

- `counter`: counter event occurred in this execution
- `end`: add to which end of the deque

bool **is_same** (const *ExecData_t* &*other*) const
compare with other execution

Return true if they are same

Return false if they are different

Parameters

- `other`: other execution data

nlohmann::json **get_json** (bool *with_message* = false) const
Get the json object of this execution data.

Return nlohmann::json json object

Parameters

- `with_message`: if true, including all message (communication) information

bool **can_delete** () const
Determine whether the event can be deleted by the garbage collection at the end of the io step.

void **can_delete** (const bool *v*)
Set whether the event can be deleted by the garbage collection at the end of the io step (default true)

void **set_GPU_correlationID_partner** (const std::string *event_id*)
Set the partner event linked by a GPU correlation ID.

bool **has_GPU_correlationID_partner** () const
Return true if this event has been matched to a partner event by a GPU correlation ID.

const std::string &**get_GPU_correlationID_partner** () const
Get the partner event linked by a GPU correlation ID (empty string if none)

Private Members

std::string **m_id**
execution id

std::string **m_funcname**
function name

unsigned long **m_pid**
program id

unsigned long **m_tid**
thread id

unsigned long **m_rid**
rank id

unsigned long **m_fid**
function id

long **m_entry**
entry time

long **m_exit**
exit time


```

long m_runtime
    inclusive running time (i.e. including time of child calls)

long m_exclusive
    exclusive running time (i.e. excluding time of child calls)

int m_label
    1 for normal, -1 for abnormal execution

std::string m_parent
    parent execution

unsigned long m_n_children
    the number of childrent executions

unsigned long m_n_messages
    the number of messages

std::deque<CommData_t> m_messages
    a vector of all messages

std::deque<CounterData_t> m_counters
    a vector of all counters

bool m_can_delete
    Flag indicating that the event is deletable by the garbage collection

std::string m_gpu_correlation_id_partner
    The event id of a partner event linked by a correlation ID, either the launching CPU event or the GPU
    kernel event

class MetaData_t
    #include <ExecData.hpp> wrapper for metadata entries

```

Public Functions

```

MetaData_t (unsigned long rank, unsigned long tid, const std::string &descr, const std::string
    &value)
    Construct an instance will full set of parameters.

unsigned long get_comm_rank () const
    Get the origin global comm rank.

unsigned long get_tid () const
    Get the origin thread index.

const std::string &get_descr () const
    Get the metadata description.

const std::string &get_value () const
    Get the metadata value.

nlohmann::json get_json () const
    Get the json object of this metadata.

Return nlohmann::json json object

```

Private Members

unsigned long **m_rank**
Global comm rank

unsigned long **m_tid**
Thread idx

std::string **m_descr**
Metadata description

std::string **m_value**
Metadata value

utils

namespace chimbuko

Functions

unsigned char **random_char** ()
Return a random character.

std::string **generate_hex** (const unsigned int *len*)

std::string **generate_event_id** (int *rank*, int *step*, size_t *idx*)

std::string **generate_event_id** (int *rank*, int *step*, size_t *idx*, unsigned long *eid*)

6.1.2 Anomaly Detection Algorithm Parameters

Parameters of the anomaly detection algorithm.

ParamInterface

namespace chimbuko

class **NetPayloadGetParams** : public *chimbuko::NetPayloadBase*
#include <param.hpp> Net payload for AD updating params from pserver.

Public Functions

NetPayloadGetParams (*ParamInterface* const **param*)

MessageKind **kind** () const
The message kind to which the payload is to be bound.

MessageType **type** () const
The message type to which the payload is to be bound.

void **action** (*Message* &*response*, const *Message* &*message*)
Act on the message and formulate a response.

Private Members

ParamInterface **const** *m_param

class NetPayloadUpdateParams : **public** *chimbuko::NetPayloadBase*
#include <param.hpp> Net payload for pserver updating params from AD.

Public Functions

NetPayloadUpdateParams (*ParamInterface* *param)

MessageKind **kind** () **const**
 The message kind to which the payload is to be bound.

MessageType **type** () **const**
 The message type to which the payload is to be bound.

void **action** (*Message* &response, **const** *Message* &message)
 Act on the message and formulate a response.

Private Members

ParamInterface *m_param

class ParamInterface
#include <param.hpp> The general interface for storing function statistics for anomaly detection.
 Subclassed by *chimbuko::SstdParam*

Public Functions

ParamInterface ()

virtual ~**ParamInterface** ()

virtual **void** **clear** () = 0
 Clear all statistics.

virtual **size_t** **size** () **const** = 0
 Get the number of functions for which statistics are being collected.

virtual **std::string** **serialize** () **const** = 0
 Convert internal run statistics to string format for IO.

Return Run statistics in string format

virtual **std::string** **update** (**const** **std::string** ¶meters, **bool** flag = false) = 0
 Update the internal run statistics with those included in the serialized input map.

Return Returned contents dependent on implementation

Parameters

- parameters: The parameters in serialized format
- flag: The meaning of the flag is dependent on the implementation

```
virtual void assign (const std::string &parameters) = 0
```

Set the internal run statistics to match those included in the serialized input map. Overwrite performed only for those keys in input.

Parameters

- `runstats`: The serialized input map

```
virtual void show (std::ostream &os) const = 0
```

```
virtual const RunStats &get_function_stats (const unsigned long func_id) const = 0
```

Get the statistics associated with a given function.

Protected Attributes

```
std::mutex m_mutex
```

SstdParam

```
namespace chimbuko
```

```
class SstdParam: public chimbuko::ParamInterface
```

```
#include <sstd_param.hpp> @brief Implementation of ParamInterface for anomaly detection based on function time distribution (mean, std. dev., etc)
```

Public Functions

```
SstdParam()
```

```
~SstdParam()
```

```
void clear()
```

Clear all statistics.

```
size_t size() const
```

Get the number of functions for which statistics are being collected.

```
std::string serialize() const
```

Convert internal run statistics to string format for IO.

Return Run statistics in string format

```
std::string update (const std::string &parameters, bool return_update = false)
```

Update the internal run statistics with those included in the serialized input map.

Return An empty string if `return_update==False`, otherwise the serialized updated parameters

Parameters

- `parameters`: The parameters in serialized format
- `return_update`: Controls return format

```
void assign (const std::string &parameters)
```

Set the internal run statistics to match those included in the serialized input map. Overwrite performed only for those keys in input.

Parameters

- `runstats`: The serialized input map

void **show** (std::ostream &*os*) **const**

void **update** (**const** std::unordered_map<unsigned long, *RunStats*> &*runstats*)

Update the internal run statistics with those included in the input map.

Parameters

- [in] `runstats`: The input map

void **update** (**const** *SstdParam* &*other*)

Update the internal statistics with those included in another *SstdParam* instance.

Parameters

- [in] `other`: The other *SstdParam* instance

void **update_and_return** (std::unordered_map<unsigned long, *RunStats*> &*runstats*)

Update the internal run statistics with those included in the input map. Input map is then updated to reflect new state.

Parameters

- [inout] `runstats`: The input/output map

void **update_and_return** (*SstdParam* &*other*)

Update the internal statistics with those included in another *SstdParam* instance. Other *SstdParam* is then updated to reflect new state.

Parameters

- [inout] `other`: The other *SstdParam* instance

void **assign** (**const** std::unordered_map<unsigned long, *RunStats*> &*runstats*)

Set the internal run statistics to match those included in the input map. Overwrite performed only for those keys in input.

Parameters

- `runstats`: The input map

RunStats &**operator**[] (unsigned long *id*)

Get an element of the internal map. *id* is the function index.

const std::unordered_map<unsigned long, *RunStats*> &**get_runstats** () **const**

Get the internal map.

const *RunStats* &**get_function_stats** (**const** unsigned long *func_id*) **const**

Get the statistical distribution associated with a given function.

Public Static Functions

static std::string **serialize** (**const** std::unordered_map<unsigned long, *RunStats*> &*runstats*)

Convert a run statistics mapping into a string.

Return Run statistics in string format

Parameters

- The: run stats mapping

static void **deserialize** (**const** std::string &*parameters*, std::unordered_map<unsigned long, *RunStats*> &*runstats*)

Convert a run statistics string into a map.

Parameters

- [in] `parameters`: The parameter string
- [out] `runstats`: The run stats map

Private Members

`std::unordered_map<unsigned long, RunStats> m_runstats`
Map of function index to statistics

6.1.3 Parameter Server

The parameter server runs on the head node and aggregates function anomaly and counter statistics for visualization. Aggregated statistics for function executions are also maintained and synchronized back to the AD instances such that the anomaly detection algorithm uses the most complete statistics to identify anomalies.

global_anomaly_stats

`namespace chimbuko`

class GlobalAnomalyStats

`#include <global_anomaly_stats.hpp>` Interface for collection of global anomaly statistics on parameter server.

Public Functions

GlobalAnomalyStats()

~GlobalAnomalyStats()

GlobalAnomalyStats(const std::vector<int> &n_ranks)

Initialize global anomaly stats for a job spanning the given number of MPI ranks.

Parameters

- `n_ranks`: A vector of integers where each entry `i` gives the number of ranks for program index `i`

void reset_anomaly_stat(const std::vector<int> &n_ranks)

Clear all collected anomaly statistics and revert to initial stat.

Parameters

- `n_ranks`: A vector of integers where each entry `i` gives the number of ranks for program index `i`

void add_anomaly_data(const std::string &data)

Merge internal statistics with those contained within the JSON-formatted string 'data'.

std::string get_anomaly_stat(const std::string &stat_id) const

Get the JSON-formatted string corresponding to the anomaly statistics for a given program/rank.

Parameters

- `stat_id`: A string of the format "<PROGRAM IDX>:<RANK>" (eg "0:1" for program 0, rank 1)

size_t **get_n_anomaly_data** (const std::string &stat_id) const

Get the number of anomalies detected for a given program/rank.

Parameters

- stat_id: A string of the format “<PROGRAM IDX>:<RANK>” (eg “0:1” for program 0, rank 1)

void **update_func_stat** (unsigned long id, const std::string &name, unsigned long n_anomaly, const RunStats &inclusive, const RunStats &exclusive)

Update internal data to include additional information.

Parameters

- id: Function index
- name: Function name
- n_anomaly: The number of anomalies detected
- inclusive: Statistics on inclusive timings
- exclusive: Statistics on exclusive timings

nlohmann::json **collect_stat_data** ()

Collect anomaly statistics into JSON object and flush the m_anomaly_stats statistics.

nlohmann::json **collect_func_data** () const

Collect function statistics into JSON object.

nlohmann::json **collect** ()

Collect anomaly statistics and function statistics. Flushes the m_anomaly_stats statistics.

Return JSON object containing anomaly and function data

Protected Attributes

std::unordered_map<std::string, AnomalyStat *> **m_anomaly_stats**

Global anomaly statistics indexed by a stat_id of form “\${app_id}:\${rank_id}”

std::mutex **m_mutex_func**

std::unordered_map<unsigned long, std::string> **m_func**

Map of index to function name

std::unordered_map<unsigned long, RunStats> **m_func_anomaly**

Map of index to statistics on number of anomalies

std::unordered_map<unsigned long, RunStats> **m_inclusive**

Map of index to statistics on function timings inclusive of children

std::unordered_map<unsigned long, RunStats> **m_exclusive**

Map of index to statistics on function timings exclusive of children

class NetPayloadUpdateAnomalyStats : public chimbuko::NetPayloadBase

#include <global_anomaly_stats.hpp> Net payload for communicating anomaly stats AD->pserver.

Public Functions

NetPayloadUpdateAnomalyStats (*GlobalAnomalyStats* **global_anom_stats*)

MessageKind **kind** () **const**

The message kind to which the payload is to be bound.

MessageType **type** () **const**

The message type to which the payload is to be bound.

void action (*Message* &*response*, **const** *Message* &*message*)

Act on the message and formulate a response.

Private Members

GlobalAnomalyStats ***m_global_anom_stats**

class PSstatSenderGlobalAnomalyStatsPayload: **public** *chimbuko::PSstatSenderPayloadBase*
#include <*global_anomaly_stats.hpp*> Payload object for communicating anomaly data pserver->viz.

Public Functions

PSstatSenderGlobalAnomalyStatsPayload (*GlobalAnomalyStats* **stats*)

void add_json (*nlohmann::json* &*into*) **const**

Add the JSON object payload to 'into' as a new member with an appropriate tag (user should ensure no duplicate tags!)

Private Members

GlobalAnomalyStats ***m_stats**

global_counter_stats

namespace chimbuko

class GlobalCounterStats

#include <*global_counter_stats.hpp*> Interface for collection of global counter statistics on parameter server.

Public Functions

void add_data (**const** *std::string* &*data*)

Merge internal statistics with those contained within the JSON-formatted string 'data'.

For data format see *ADLocalCounterStatistics::get_json_state()*

std::unordered_map<*std::string*, *RunStats*> **get_stats** () **const**

Return a copy of the internal counter statistics.

nlohmann::json **get_json_state** () **const**

Serialize the state into a JSON object for sending to viz.

Protected Attributes

std::mutex **m_mutex**

std::unordered_map<std::string, *RunStats*> **m_counter_stats**
Map of counter name to global statistics

class NetPayloadUpdateCounterStats : public *chimbuko::NetPayloadBase*
#include <global_counter_stats.hpp> Net payload for communicating counter stats AD->pserver.

Public Functions

NetPayloadUpdateCounterStats (*GlobalCounterStats* **global_counter_stats*)

MessageKind **kind** () **const**
The message kind to which the payload is to be bound.

MessageType **type** () **const**
The message type to which the payload is to be bound.

void **action** (*Message* &*response*, **const** *Message* &*message*)
Act on the message and formulate a response.

Private Members

GlobalCounterStats ***m_global_counter_stats**

class PSstatSenderGlobalCounterStatsPayload : public *chimbuko::PSstatSenderPayloadBase*
#include <global_counter_stats.hpp> Payload object for communicating counter data pserver->viz.

Public Functions

PSstatSenderGlobalCounterStatsPayload (*GlobalCounterStats* **stats*)

void **add_json** (nlohmann::json &*into*) **const**
Add the JSON object payload to 'into' as a new member with an appropriate tag (user should ensure no duplicate tags!)

Private Members

GlobalCounterStats ***m_stats**

PSglobalFunctionIndexMap

namespace **chimbuko**

class NetPayloadGlobalFunctionIndexMap : public *chimbuko::NetPayloadBase*
#include <PSglobalFunctionIndexMap.hpp> Net payload for communicating function index pserver->AD.

Public Functions

NetPayloadGlobalFunctionIndexMap (*PSglobalFunctionIndexMap* *idxmap)

MessageKind **kind** () **const**

The message kind to which the payload is to be bound.

MessageType **type** () **const**

The message type to which the payload is to be bound.

void action (*Message* &response, **const** *Message* &message)

Act on the message and formulate a response.

Private Members

PSglobalFunctionIndexMap ***m_idxmap**

class PSglobalFunctionIndexMap

#include <PSglobalFunctionIndexMap.hpp> A class that maintains a global mapping between function name and an index, which is to be synchronized over the nodes.

Public Functions

unsigned long lookup (**const** std::string &func_name)

Lookup a function by name and return the index. A new index will be assigned if the function has not been encountered before.

Private Members

std::unordered_map<std::string, unsigned long> **m_fmap**

std::mutex **m_mutex**

PSstatSender

namespace chimbuko

class PSstatSender

#include <PSstatSender.hpp> A class that periodically sends aggregate statistics to the visualization module via curl using a background thread.

Public Functions

PSstatSender (size_t send_freq = 1000)

Constructpr.

Parameters

- **send_freq**: The frequency (in milliseconds) at which sends are performed to the viz module

~PSstatSender ()

void **set_send_freq** (const size_t *freq*)

Change the frequency (in milliseconds) at which sends are performed to the viz module. Must be set prior to calling run_stat_sender.

void **run_stat_sender** (std::string *url*)

Start sending global anomaly stats to the visualization module (curl)

Parameters

- *url*: The URL of the visualization module

void **stop_stat_sender** (int *wait_msec* = 0)

Stop sending global anomaly stats to the visualization module (curl)

void **add_payload** (*PSstatSenderPayloadBase* **payload*)

Add a payload. Takes ownership of pointer, which is freed.

bool **bad** () const

If an exception is caught in the thread loop, the thread will stop issuing sends and set this bool to true.

Private Members

size_t **m_send_freq**

Number of seconds between sends to viz

std::thread ***m_stat_sender**

std::atomic_bool **m_stop_sender**

std::atomic_bool **m_bad**

If an exception is caught in the thread loop, the thread will stop issuing sends and set this bool to true

std::vector<*PSstatSenderPayloadBase* *> **m_payloads**

Vector of payload wrappers defining the sets of data sent to the parameter server

struct PSstatSenderPayloadBase

#include <PSstatSender.hpp> Base class for wrappers around objects/object pointers that return JSON objects that are sent to the parameter server.

The JSON objects are collected into a single object whose members are tagged according to the “tag” provided by the wrapper Nothing will be sent if the resulting JSON object is empty

Subclassed by *chimbuko::PSstatSenderGlobalAnomalyStatsPayload*, *chimbuko::PSstatSenderGlobalCounterStatsPayload*

Public Functions

virtual void **add_json** (nlohmann::json &*into*) const = 0

Add the JSON object payload to ‘into’ as a new member with an appropriate tag (user should ensure no duplicate tags!)

virtual bool **do_fetch** () const

Whether to request a callback to process the response (optional)

Parameters

- *packet*: The string packet returned by the previous call to get_json()
- *returned*: The string returned in response

```
virtual void process_callback (const std::string &packet, const std::string &returned)
                                const
    If a callback is requested, this function is called after it is returned.

virtual ~PSstatSenderPayloadBase ()
```

6.1.4 Network

The network is the communication pathway between the AD instances and the parameter server. The default implementation, ZMQnet uses zeroMQ, and a deprecated interface via MPI is also provided and can be selected at compile time.

NetInterface

```
namespace chimbuko
```

Enums

```
enum NetThreadLevel
    enum network thread level (for MPI)

    Values:

    THREAD_MULTIPLE = 3
```

```
class NetInterface
    #include <net.hpp> Network interface class.
    Subclassed by chimbuko::ZMQNet
```

Public Functions

```
NetInterface ()
    Construct a new Net Interface object.
```

```
virtual ~NetInterface ()
    Destroy the Net Interface object.
```

```
virtual void init (int *argc = nullptr, char ***argv = nullptr, int nt = 1) = 0
    (virtual) initialize network interface
```

Parameters

- *argc*: command line argc
- *argv*: command line argv
- *nt*: the number of threads for a thread pool

```
virtual void finalize () = 0
    (virtual) finalize network
```

```
virtual void run () = 0
    (virtual) run network server
```

```
virtual void stop () = 0
    (virtual) stop network server
```

virtual std::string **name** () **const** = 0

(virtual) name of network server

Return std::string name of network server

void **add_payload** (*NetPayloadBase* *payload)

Add a payload to the receiver bound to particular message kind/type specified internally.

Assumes ownership of the *NetPayloadBase* object and deletes in constructor

Protected Functions

virtual void **init_thread_pool** (int nt) = 0

initialize thread pool

Parameters

- nt: the number threads in the pool

Protected Attributes

int **m_nt**

The number of threads in the pool

std::unordered_map<*MessageKind*, std::unordered_map<*MessageType*, std::unique_ptr<*NetPayloadBase*>>> **m_payload**

class NetPayloadBase

Subclassed by *chimbuko::NetPayloadGetParams*, *chimbuko::NetPayloadGlobalFunctionIndexMap*, *chimbuko::NetPayloadHandShake*, *chimbuko::NetPayloadUpdateAnomalyStats*, *chimbuko::NetPayloadUpdateCounterStats*, *chimbuko::NetPayloadUpdateParams*

Public Functions

virtual *MessageKind* **kind** () **const** = 0

The message kind to which the payload is to be bound.

virtual *MessageType* **type** () **const** = 0

The message type to which the payload is to be bound.

virtual void **action** (*Message* &response, **const** *Message* &message) = 0

Act on the message and formulate a response.

void **check** (**const** *Message* &msg) **const**

Helper function to ensure the message is of the correct kind/type.

virtual ~**NetPayloadBase** ()

class **NetPayloadHandShake** : **public** *chimbuko::NetPayloadBase*

#include <net.hpp> Default handshake response; this is bound automatically to the network.

Public Functions

MessageKind **kind** () **const**

The message kind to which the payload is to be bound.

MessageType **type** () **const**

The message type to which the payload is to be bound.

void **action** (*Message* &*response*, **const** *Message* &*message*)

Act on the message and formulate a response.

namespace DefaultNetInterface

Functions

NetInterface &**get** ()

get default network interface for easy usages

Return *NetInterface*& default network

MPINet

ZMQNet

namespace chimbuko

class ZMQNet : **public** *chimbuko::NetInterface*

#include <zmq_net.hpp> A network interface using ZeroMQ.

Public Functions

ZMQNet ()

~ZMQNet ()

void **init** (int **argc*, char ****argv*, int *nt*)

(virtual) initialize network interface

Parameters

- *argc*: command line argc
- *argv*: command line argv
- *nt*: the number of threads for a thread pool

void **finalize** ()

(virtual) finalize network

void **run** ()

(virtual) run network server

void **stop** ()

(virtual) stop network server

std::string **name** () **const**

(virtual) name of network server

Return std::string name of network server

Public Static Functions

static int send (void *socket, **const** std::string &strmsg)

static int recv (void *socket, std::string &strmsg)

Protected Functions

void **init_thread_pool** (int nt)

initialize thread pool

Parameters

- nt: the number threads in the pool

Private Functions

bool **recvAndSend** (void *skFrom, void *skTo)

Route a message to/from worker thread pool.

Private Members

void ***m_context**

ZeroMQ context pointer

long long **m_n_requests**

std::vector<std::thread> **m_threads**

The pool of thread workers

6.1.5 Message

namespace chimbuko

Enums

enum **MessageType**

Values:

REQ_ADD = 1

REQ_GET = 2

REQ_CMD = 3

REQ_QUIT = 4

REQ_ECHO = 5

REP_ADD = 10

REP_GET = 20

REP_CMD = 30

```
    REP_QUIT = 40
    REP_ECHO = 50
enum MessageKind
    Values:
    DEFAULT = 0
    CMD = 1
    PARAMETERS = 2
    ANOMALY_STATS = 3
    COUNTER_STATS = 4
    FUNCTION_INDEX = 5
enum MessageCmd
    Values:
    QUIT = 0
    ECHO = 1
class Message
```

Public Functions

Message ()

Construct a new *Message* object.

~Message ()

Destroy the *Message* object.

void **set_info** (int *src*, int *dst*, int *type*, int *kind*, int *frame* = 0, int *size* = 0)

Set the message information (header)

Parameters

- *src*: source rank
- *dst*: destination rank
- *type*: message type
- *kind*: message kind
- *frame*: frame index
- *size*: message size

void **set_msg** (const std::string &*msg*, bool *include_head* = false)

Set the message contents.

If 'include_head' is true, the string 'msg' will be interpreted as a JSON object and the '*Header*' field will be used to fill the header portion of the message and the 'Buffer' field as the contents. If 'include_head' is false, the message contents will be set to 'msg' and the header will be set to contain the length of the string as its size entry.

void **set_msg** (int *cmd*)

Set the message contents to an integer; equivalent to set_msg(int_as_string, false)

const std::string &**buf** () **const**

Return the message contents as a stringized JSON object containing the '*Header*' and 'Buffer' fields corresponding to the header and message contents, resp.


```
std::string data () const
```

Return the message as a stringized JSON object containing the header and contents.

```
int src () const
```

```
int dst () const
```

```
int type () const
```

```
int kind () const
```

```
std::string kind_str () const
```

```
int size () const
```

```
int frame () const
```

```
void clear ()
```

clear data buffer

```
Message createReply () const
```

```
void show (std::ostream &os) const
```

Private Members

```
Header m_head
```

```
std::string m_buf
```

```
class Header
```

Public Functions

```
Header ()
```

header size in bytes

```
int &src ()
```

source rank

Return int& reference to the source rank

```
int src () const
```

```
int &dst ()
```

desination rank

Return int& reference to the destination rank

```
int dst () const
```

```
int &type ()
```

message type

Return int& reference to the message type

```
int type () const
```

```
int &kind()  
    message kind  
  
    Return int& reference to the message kind  
  
int kind() const  
  
int &size()  
    message size  
  
    Return int& reference to the message size  
  
int size() const  
  
int &frame()  
    message frame index  
  
    Return int& reference to the message frame index  
  
int frame() const  
  
nlohmann::json get_json() const  
  
void set_header(const nlohmann::json &j)  
  
void set_header(const std::string &s)
```

Private Members

```
int m_h[8]  
    header information  
  
    0: src rank 1: dst rank 2: message type 3: message kind 4: message size (except header) in bytes  
    5: frame index (or step index) 6: reserved 7: reserved
```

6.1.6 Utils

Utility functions and classes.

ADIOS2parseUtils

namespace chimbuko

Functions

```
std::ostream &operator<< (std::ostream &os, const mapPrint &mp)  
    ostream output of a map using mapPrint wrapper  
  
template<typename T>  
std::ostream &operator<< (std::ostream &os, const vecPrint<T> &mp)  
    ostream output of a vector using vecPrint wrapper
```

varBase ***parseVariable** (**const** std::string &*name*, **const** std::map<std::string, std::string> &*varinfo*, adios2::IO &*io*, adios2::Engine &*eng*)

A factory for generating *varBase* derived class instances that contain the data read from the input stream.

Returns a NULL ptr if the type is not supported The name/varinfo data can be obtained using the adios2::IO::AvailableVariables method

struct mapPrint

#include <ADIOS2parseUtils.hpp> Wrapper allowing ostream output of a string map object.

Public Functions

mapPrint (**const** std::map<std::string, std::string> &*mp*)

Public Members

const std::map<std::string, std::string> &*mp*

struct varBase

#include <ADIOS2parseUtils.hpp> Abstract interface for an object that reads, stores and outputs data or arrays of data from ADIOS2 streams.

Subclassed by *chimbuko::varPOD< T >*, *chimbuko::varTensor< T >*

Public Functions

varBase (**const** std::string &*name*)

Construct object with variable name 'name'.

virtual std::string **value** () **const**

Get the value as a human-readable string.

virtual void **get** (adios2::IO &*io*, adios2::Engine &*eng*)

Read the variable from the ADIOS2 stream.

virtual void **put** (adios2::IO &*io*, adios2::Engine &*eng*)

Write the variable to the ADIOS2 stream.

virtual ~**varBase** ()

Public Members

std::string **name**

template<typename T>

class **varPOD** : **public** *chimbuko::varBase*

#include <ADIOS2parseUtils.hpp> Capture POD (single-value) data.

Public Functions

```
varPOD (const std::string &name)

varPOD (const std::string &name, adios2::IO &io, adios2::Engine &eng)

void get (adios2::IO &io, adios2::Engine &eng)
    Read the variable from the ADIOS2 stream.

virtual void put (adios2::IO &io, adios2::Engine &eng)
    Write the variable to the ADIOS2 stream.

std::string value () const
    Get the value as a human-readable string.
```

Private Members

```
T val

template<typename T>
class varTensor : public chimbuco::varBase
    #include <ADIOS2parseUtils.hpp> Capture multi-dimensional tensor data.
```

Public Functions

```
varTensor (const std::string &name)

varTensor (const std::string &name, const std::vector<unsigned long> &shape, adios2::IO
    &io, adios2::Engine &eng)

void get (adios2::IO &io, adios2::Engine &eng)
    Read the variable from the ADIOS2 stream.

void put (adios2::IO &io, adios2::Engine &eng)
    Write the variable to the ADIOS2 stream.

std::string value () const
    Get the value as a human-readable string.

T &operator () (const std::vector<unsigned long> &coord)
    Get the value at given coordinate (non-const)

const T &operator () (const std::vector<unsigned long> &coord) const
    Get the value at given coordinate (const)

const std::vector<unsigned long> &getShape () const
    Get the shape of the tensor.
```

Private Functions

```
template<typename listType>
size_t map (const listType &c) const
    Compute the lexicographic offset for coordinate 'c' assuming row-major order.

void unmap (std::vector<unsigned long> &c, size_t o) const
    Unmap an offset into a coordinate.
```

Private Members

```
std::vector<unsigned long> shape
    The "shape" of the tensor

std::vector<T> val

template<typename T>
struct vecPrint
    #include <ADIOS2parseUtils.hpp> Wrapper allowing ostream output of a vector object.
```

Public Functions

```
vecPrint (const std::vector<T> &mp)
```

Public Members

```
const std::vector<T> &mp
```

Anomalies

```
namespace chimbuko
```

```
class Anomalies
    #include <Anomalies.hpp> A class that contains information about the anomalies captured by the AD.
    Also stored are a few examples of normal executions, allowing for comparison with outliers.
```

Public Types

```
enum EventType
    Values:

    Outlier

    Normal
```

Public Functions

void **insert** (*CallListIterator_t* event, *EventType* type)
Insert a detected outlier/normal execution.

const std::vector<*CallListIterator_t*> &**funcEvents** (**const** unsigned long *func_id*, *EventType*
type) **const**
Get the outlier/normal events associated with a given function.

const std::vector<*CallListIterator_t*> &**allEvents** (*EventType* type) **const**
Get all outliers/normal events.

size_t **nFuncEvents** (**const** unsigned long *func_id*, *EventType* type) **const**
Get number of outliers/normal events associated with a given function.

size_t **nEvents** (*EventType* type) **const**
Get number of outliers/normal events.

Private Members

std::vector<*CallListIterator_t*> **m_all_outliers**
Array of outliers

std::unordered_map<unsigned long, std::vector<*CallListIterator_t*>> **m_func_outliers**
Map of function index to associated outliers

std::vector<*CallListIterator_t*> **m_all_normal_execs**
Array of normal executions (the algorithm will capture a limited number of these for comparison with outliers)

std::unordered_map<unsigned long, std::vector<*CallListIterator_t*>> **m_func_normal_execs**
Map of function index to associated normal executions

barrier

namespace chimbuko

class Barrier
#include <barrier.hpp> Thread barrier.

Public Functions

Barrier (std::size_t *iCount*)
Constructor.

Parameters

- *iCount*: The number of threads in the barrier

void **wait** ()

Private Members

```
std::mutex mMutex
std::condition_variable mCond
std::size_t mThreshold
std::size_t mCount
std::size_t mGeneration
```

commandLineParser

Defines

addCommandLineArg (PARSER, NAME, HELP_STR)

Helper macro to add a command line arg to the parser PARSER with given name NAME and help string HELP_STR.

addCommandLineArgDefaultHelpString (PARSER, NAME)

Helper macro to add a command line arg to the parser PARSER with given name NAME and default help string “Provide the value for NAME”.

namespace chimbuko

```
template<typename ArgsStruct, typename T, T ArgsStruct::*P>
class commandLineArg : public chimbuko::commandLineArgBase<ArgsStruct>
    #include <commandLineParser.hpp> A class that parses an argument of a given type into the struct.
```

Public Functions

commandLineArg (**const** std::string &*arg*, **const** std::string &*help_str*)
Create an instance with the provided argument and help string.

bool **parse** (**ArgsStruct** &*into*, **const** std::string &*arg*, **const** std::string &*val*)
If the first string matches the internal arg string (eg “-help”) parse the second string *val* and return true. If first string doesn’t match or *val* is unable to be parsed, return false.

void **help** (std::ostream &*os*) **const**
Print the help string for this argument to the ostream.

Private Members

```
std::string m_arg
    The argument, format “-a”
std::string m_help_str
    The help string
```

```
template<typename ArgsStruct>
class commandLineArgBase
    #include <commandLineParser.hpp> Base class for arg parsing structs.
    Subclassed by chimbuko::commandLineArg< ArgsStruct, T, P >
```

Public Functions

virtual bool **parse** (ArgsStruct &into, **const** std::string &arg, **const** std::string &val) = 0

If the first string matches the internal arg string (eg “-help”) parse the second string val and return true. If first string doesn’t match or val is unable to be parsed, return false.

virtual void **help** (std::ostream &os) **const** = 0

Print the help string for this argument to the ostream.

virtual ~commandLineArgBase ()

template<typename **ArgsStruct**>

class **commandLineParser**

#include <commandLineParser.hpp> The main parser class for a generic struct ArgsStruct.

Public Types

typedef ArgsStruct **StructType**

Public Functions

template<typename **T**, *T* **ArgsStruct**::*P>

void **addArg** (**const** std::string &arg, **const** std::string &help_str)

Add an argument with the given type, member pointer (eg &ArgsStruct::a) with provided argument (eg “-a”) and help string.

void **parse** (ArgsStruct &into, **const** int narg, **const** char **args)

Parse an array of strings of length ‘narg’ into the structure.

void **help** (std::ostream &os = std::cout) **const**

Print the help information for all the args that can be parsed.

Private Members

std::vector<std::unique_ptr<commandLineArgBase<ArgsStruct>>> **m_args**

Container for the individual arg parsers

DispatchQueue

namespace **chimbuko**

class **DispatchQueue**

#include <DispatchQueue.hpp> A class for dispatching work items over a thread pool.

Public Functions

DispatchQueue (std::string *name*, size_t *thread_cnt* = 1)

Construct an instance of class, providing a name for the instance and the number of threads.

Parameters

- *name*: The name of the instance
- *thread_cnt*: The number of threads (default 1)

~DispatchQueue ()

void **dispatch** (const *fp_t* &*op*)

Enqueue a work item (lvalue reference)

Parameters

- *op*: An instance of std::function<void(void)>

void **dispatch** (*fp_t* &&*op*)

Enqueue a work item (rvalue reference)

Parameters

- *op*: An instance of std::function<void(void)>

size_t **size** ()

Return the number of outstanding work items in the queue.

Private Types

typedef std::function<void (void) > **fp_t**

Private Functions

void **thread_handler** (void)

Private Members

std::string **m_name**

std::mutex **m_lock**

std::vector<std::thread> **m_threads**

std::queue<*fp_t*> **m_q**

std::condition_variable **m_cv**

bool **m_quit**

hash

namespace chimbuko

```
template<typename T, size_t N>
struct ArrayHasher
    #include <hash.hpp> Hash function for std::array.
```

Public Functions

```
std::size_t operator () (const std::array<T, N> &a) const
```

mtQueue

```
template<typename T>
class mtQueue
```

Public Functions

```
mtQueue ()
~mtQueue ()

bool tryPop (T &out)
bool waitPop (T &out)
void push (T value)
bool empty () const
    Return true if the queue is empty.
void clear ()
    Remove all entries from the queue.
void invalidate ()
bool is_valid () const
size_t size () const
    The number of entries in the queue.
```

Private Members

```
std::atomic_bool m_valid = {true}
std::mutex m_mutex
std::queue<T> m_queue
std::condition_variable m_cond
```

PerfStats

```
namespace chimbuko
```

class PerfStats

#include <PerfStats.hpp> A class that maintains performance statistics of various aspects of the AD module. Its constituent functions only do anything if `_PERF_METRIC` flag is enabled.

Public Functions

```
PerfStats ()
```

```
PerfStats (const std::string &output_path, const std::string &filename)
```

```
void add (const std::string &label, const double value)
```

```
void setWriteLocation (const std::string &output_path, const std::string &filename)
```

Set the output path and file name.

```
void write () const
```

Write the running statistics to the file. Only writes out if a path and filename have been provided.

class PerfTimer

#include <PerfStats.hpp> A timer class that only measures time if `_PERF_METRIC` compile flag is set.

Public Functions

```
PerfTimer (bool start_now = true)
```

```
void start ()
```

(Re)start the timer

```
double elapsed_us () const
```

Compute the elapsed time in microseconds since start.

```
double elapsed_ms () const
```

Compute the elapsed time in milliseconds since start.

RunMetric

```
namespace chimbuko
```

class RunMetric

Public Functions

```
RunMetric()  
~RunMetric()  
  
void add(std::string name, double val)  
  
void dump(std::string path, std::string filename = "metric.json") const
```

Private Members

```
std::unordered_map<std::string, RunStats> m_metrics
```

RunStats

```
namespace chimbuko
```

Functions

```
RunStats operator+(const RunStats a, const RunStats b)  
  
bool operator==(const RunStats &a, const RunStats &b)  
  
bool operator!=(const RunStats &a, const RunStats &b)  
  
double static_mean(const std::vector<double> &data, double ddof = 1.0)  
  
double static_std(const std::vector<double> &data, double ddof = 1.0)
```

class RunStats

```
#include <RunStats.hpp> Compute statistics in a single pass.
```

Computes the minimum, maximum, mean, variance, standard deviation, skewness, and kurtosis. Optionally, also computes accumulated values.

RunStats objects may also be added together and copied.

Based entirely on the C++ code by John D Cook at http://www.johndcook.com/skewness_kurtosis.html

Public Types

```
typedef struct chimbuko::RunStats::State State  
    Internal state of RunStats object.
```

Public Functions

```

RunStats (bool do_accumulate = false)

~RunStats ()

void clear ()

State get_state ()

void set_state (const State &s)

RunStats copy ()

void set_json_state (const nlohmann::json &s)

std::string get_strstate ()

void set_strstate (const std::string &s)

void push (double x)
    Add a new value to be included in internal statistics.

double count () const

double minimum () const

double maximum () const

double accumulate () const

double mean () const

double variance (double ddof = 1.0) const

double stddev (double ddof = 1.0) const

double skewness () const

double kurtosis () const

void set_do_accumulate (bool do_accumulate)

nlohmann::json get_json () const

nlohmann::json get_json_state () const

RunStats &operator+= (const RunStats &rs)

```

Public Static Functions

```

static RunStats from_state (const State &s)

static RunStats from_json_state (const nlohmann::json &s)

static RunStats from_strstate (const std::string &s)

```

Private Members

State **m_state**

bool **m_do_accumulate**

Friends

RunStats **operator+** (const RunStats *a*, const RunStats *b*)

bool **operator==** (const RunStats &*a*, const RunStats &*b*)

bool **operator!=** (const RunStats &*a*, const RunStats &*b*)

struct State

#include <RunStats.hpp> Internal state of *RunStats* object.

Public Functions

State ()

State (double *_count*, double *_eta*, double *_rho*, double *_tau*, double *_phi*, double *_min*, double *_max*, double *_acc*)

void **clear** ()

Public Members

double **count**
count of instances

double **eta**
mean

double **rho**

double **tau**

double **phi**

double **min**
minimum

double **max**
maximum

double **acc**

string

namespace chimbuko

Functions

```
template<typename T>
T strToAny (const std::string &s)
    Convert string to anything.

template<>
std::string strToAny<std::string> (const std::string &s)

template<typename T>
std::string anyToStr (const T &v)
    Convert any type to string.

template<>
std::string anyToStr<std::string> (const std::string &s)

std::string stringize (const char *format, ...)
    C-style string formatting but without the nasty mem buffer concerns.
```

threadPool

class threadPool

Public Functions

```
threadPool ()

threadPool (const std::uint32_t nt)
    Instantiate a pool of nt threads.
```

Parameters

- nt: The number of threads to instantiate

```
threadPool (const threadPool &rhs)
    The class is not copyable but can be moved.

threadPool &operator= (const threadPool &rhs)

~threadPool ()

template<typename Func, typename ...Args>
auto submit (Func &&func, Args&&... args)

size_t pool_size () const

size_t queue_size () const
```

Private Functions

void **worker** ()

void **destroy** ()

Private Members

std::atomic_bool **m_done**

mtQueue<std::unique_ptr<*IThreadTask*>> **m_workQueue**

std::vector<std::thread> **m_threads**

class IThreadTask

Public Functions

IThreadTask ()

virtual ~**IThreadTask** ()

IThreadTask (const *IThreadTask* &rhs)

IThreadTask &**operator=** (const *IThreadTask* &rhs)

IThreadTask (*IThreadTask* &&other)

IThreadTask &**operator=** (*IThreadTask* &&other)

virtual void **execute** () = 0

template<typename T>

class TaskFuture

#include <threadPool.hpp> A wrapper class for an std::future instance representing the result of an asynchronous operation.

Public Functions

TaskFuture (std::future<T> &&*future*)

~**TaskFuture** ()

The destructor waits for the asynchronous operation to complete before exiting.

TaskFuture (const *TaskFuture* &rhs)

TaskFuture &**operator=** (const *TaskFuture* &rhs)

TaskFuture (*TaskFuture* &&other)

TaskFuture &**operator=** (*TaskFuture* &&other)

auto **get** ()

Wait until the asynchronous operation has completed and return the value.

Private Members

```
std::future<T> m_future
```

```
template<typename Func>
class ThreadTask : public threadPool::IThreadTask
```

Public Functions

```
ThreadTask (Func &&func)

~ThreadTask ()

ThreadTask (const ThreadTask &rhs)

ThreadTask &operator= (const ThreadTask &rhs)

ThreadTask (ThreadTask &&other)

ThreadTask &operator= (ThreadTask &&other)

void execute ()
```

Private Members

```
Func m_func
```

```
namespace DefaultThreadPool
```

Functions

```
threadPool &getThreadPool ()

template<typename Func, typename ...Args>
auto submitJob (Func &&func, Args&&... args)
```

verbose

Defines

```
VERBOSE (STATEMENT)
```

Macro enclosing a statement that is to only be printed if verbose mode is active.

```
namespace chimbuko
```

```
class Verbose
    #include <verbose.hpp> Static class to control verbose output.
```

Public Static Functions

static void **set_verbose** (bool *val*)

Set verbose flag.

Parameters

- *val*: The value

static bool **on** ()

Determine if verbose mode is activated.

Return Bool indicating whether verbose mode is active

Private Static Functions

static bool &**vrb** ()

Access static verbose static bool.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

INDEX

A

addCommandLineArg (*C macro*), 75

addCommandLineArgDefaultHelpString (*C macro*), 75

C

chimbuko (*C++ type*), 15, 18, 19, 22, 23, 27–29, 31–33, 35, 37, 42, 45, 54, 56, 58, 60–62, 64, 66, 67, 70, 73–76, 78–80, 83, 85

chimbuko::ADAnomalyProvenance (*C++ class*), 18

chimbuko::ADAnomalyProvenance::ADAnomalyProvenance (*C++ function*), 18

chimbuko::ADAnomalyProvenance::get_json (*C++ function*), 18

chimbuko::ADAnomalyProvenance::m_call (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_callstack (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_counters (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_func_stats (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_gpu_event_parent_info (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_gpu_location (*C++ member*), 19

chimbuko::ADAnomalyProvenance::m_is_gpu_event (*C++ member*), 19

chimbuko::ADCounter (*C++ class*), 19

chimbuko::ADCounter::~~ADCounter (*C++ function*), 20

chimbuko::ADCounter::ADCounter (*C++ function*), 20

chimbuko::ADCounter::addCounter (*C++ function*), 20

chimbuko::ADCounter::flushCounters (*C++ function*), 20

chimbuko::ADCounter::getCounters (*C++ function*), 20

chimbuko::ADCounter::getCountersByIndex (*C++ function*), 20

chimbuko::ADCounter::getCountersInWindow (*C++ function*), 20

chimbuko::ADCounter::linkCounterMap (*C++ function*), 20

chimbuko::ADCounter::m_counterMap (*C++ member*), 20

chimbuko::ADCounter::m_counters (*C++ member*), 20

chimbuko::ADCounter::m_countersByIdx (*C++ member*), 20

chimbuko::ADCounter::m_timestampCounterMap (*C++ member*), 20

chimbuko::ADEvent (*C++ class*), 24

chimbuko::ADEvent::~~ADEvent (*C++ function*), 24

chimbuko::ADEvent::addCall (*C++ function*), 26

chimbuko::ADEvent::addComm (*C++ function*), 25

chimbuko::ADEvent::addCounter (*C++ function*), 26

chimbuko::ADEvent::addEvent (*C++ function*), 25

chimbuko::ADEvent::addFunc (*C++ function*), 25

chimbuko::ADEvent::ADEvent (*C++ function*), 24

chimbuko::ADEvent::checkAndMatchCorrelationID (*C++ function*), 26

chimbuko::ADEvent::clear (*C++ function*), 25

chimbuko::ADEvent::getCallData (*C++ function*), 25

chimbuko::ADEvent::getCallListMap (*C++ function*), 25

chimbuko::ADEvent::getCounterMap (*C++ function*), 25

chimbuko::ADEvent::getEventType (*C++ function*), 25

chimbuko::ADEvent::getExecDataMap (*C++ function*), 25

chimbuko::ADEvent::getFuncMap (*C++ function*), 25

```

chimbuko::ADEvent::getUnmatchCorrelationID (C++ function), 26
chimbuko::ADEvent::linkCounterMap (C++ function), 24
chimbuko::ADEvent::linkEventType (C++ function), 24
chimbuko::ADEvent::linkFuncMap (C++ function), 24
chimbuko::ADEvent::m_callIDMap (C++ member), 27
chimbuko::ADEvent::m_callList (C++ member), 27
chimbuko::ADEvent::m_callStack (C++ member), 26
chimbuko::ADEvent::m_commStack (C++ member), 26
chimbuko::ADEvent::m_counterMap (C++ member), 26
chimbuko::ADEvent::m_counterStack (C++ member), 26
chimbuko::ADEvent::m_eventType (C++ member), 26
chimbuko::ADEvent::m_execDataMap (C++ member), 27
chimbuko::ADEvent::m_funcMap (C++ member), 26
chimbuko::ADEvent::m_unmatchedCorrelationID (C++ member), 27
chimbuko::ADEvent::m_verbose (C++ member), 27
chimbuko::ADEvent::show_status (C++ function), 26
chimbuko::ADEvent::trimCallList (C++ function), 26
chimbuko::ADglobalFunctionIndexMap (C++ class), 27
chimbuko::ADglobalFunctionIndexMap::ADglobalFunctionIndexMap (C++ function), 27
chimbuko::ADglobalFunctionIndexMap::connectedTo (C++ function), 27
chimbuko::ADglobalFunctionIndexMap::getNetClient (C++ function), 27
chimbuko::ADglobalFunctionIndexMap::linkNetClient (C++ function), 27
chimbuko::ADglobalFunctionIndexMap::lookup (C++ function), 27
chimbuko::ADglobalFunctionIndexMap::m_idxmap (C++ member), 28
chimbuko::ADglobalFunctionIndexMap::m_net_client (C++ member), 28
chimbuko::ADio (C++ class), 28
chimbuko::ADio::_open (C++ function), 29
chimbuko::ADio::~ADio (C++ function), 28
chimbuko::ADio::ADio (C++ function), 28
chimbuko::ADio::close_curl (C++ function), 28
chimbuko::ADio::destructor_thread_waittime (C++ member), 29
chimbuko::ADio::getCURL (C++ function), 28
chimbuko::ADio::getNumIOJobs (C++ function), 28
chimbuko::ADio::getOutputPath (C++ function), 28
chimbuko::ADio::getRank (C++ function), 28
chimbuko::ADio::getURL (C++ function), 28
chimbuko::ADio::getWinSize (C++ function), 28
chimbuko::ADio::m_curl (C++ member), 29
chimbuko::ADio::m_dispatcher (C++ member), 29
chimbuko::ADio::m_execWindow (C++ member), 29
chimbuko::ADio::m_outputPath (C++ member), 29
chimbuko::ADio::m_rank (C++ member), 29
chimbuko::ADio::m_url (C++ member), 29
chimbuko::ADio::open_curl (C++ function), 28
chimbuko::ADio::setDestructorThreadWaitTime (C++ function), 29
chimbuko::ADio::setDispatcher (C++ function), 28
chimbuko::ADio::setOutputPath (C++ function), 28
chimbuko::ADio::setRank (C++ function), 28
chimbuko::ADio::setWinSize (C++ function), 28
chimbuko::ADio::write (C++ function), 28
chimbuko::ADio::writeCounters (C++ function), 28
chimbuko::ADio::writeMetaData (C++ function), 29
chimbuko::ADLocalCounterStatistics (C++ class), 29
chimbuko::ADLocalCounterStatistics::ADLocalCounterStatistics (C++ function), 30
chimbuko::ADLocalCounterStatistics::gatherStatistics (C++ function), 30
chimbuko::ADLocalCounterStatistics::get_json_state (C++ function), 30
chimbuko::ADLocalCounterStatistics::getStats (C++ function), 30
chimbuko::ADLocalCounterStatistics::linkPerf (C++ function), 30
chimbuko::ADLocalCounterStatistics::m_perf (C++ member), 30
chimbuko::ADLocalCounterStatistics::m_stats (C++ member), 30
chimbuko::ADLocalCounterStatistics::m_step

```

(C++ member), 30
 chimbuko::ADLocalCounterStatistics::m_which_counter (C++ member), 30
 chimbuko::ADLocalCounterStatistics::setStats (C++ function), 30
 chimbuko::ADLocalCounterStatistics::updateGlobalStats (C++ function), 30
 chimbuko::ADLocalFuncStatistics (C++ class), 31
 chimbuko::ADLocalFuncStatistics::ADLocalFuncStatistics (C++ function), 31
 chimbuko::ADLocalFuncStatistics::gatherAnomalies (C++ function), 31
 chimbuko::ADLocalFuncStatistics::gatherStats (C++ function), 31
 chimbuko::ADLocalFuncStatistics::get_json_stats (C++ function), 31
 chimbuko::ADLocalFuncStatistics::linkPerf (C++ function), 31
 chimbuko::ADLocalFuncStatistics::m_anomaly_map (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_exclusive_map (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_func (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_inclusive_map (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_max_ts (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_min_ts (C++ member), 31
 chimbuko::ADLocalFuncStatistics::m_n_anomalies (C++ member), 32
 chimbuko::ADLocalFuncStatistics::m_perf (C++ member), 32
 chimbuko::ADLocalFuncStatistics::m_step (C++ member), 31
 chimbuko::ADLocalFuncStatistics::updateGlobalStats (C++ function), 31, 32
 chimbuko::ADMetadataParser (C++ class), 32
 chimbuko::ADMetadataParser::addData (C++ function), 32
 chimbuko::ADMetadataParser::getGPUproperties (C++ function), 32
 chimbuko::ADMetadataParser::getGPUthreads (C++ function), 32
 chimbuko::ADMetadataParser::getGPUthreadMap (C++ function), 32
 chimbuko::ADMetadataParser::isGPUthread (C++ function), 32
 chimbuko::ADMetadataParser::m_gpu_properties (C++ member), 33
 chimbuko::ADMetadataParser::m_gpu_thread_map (C++ member), 33
 chimbuko::ADMetadataParser::parseMetadata (C++ function), 33
 chimbuko::ADNetClient (C++ class), 33
 chimbuko::ADNetClient::ADNetClient (C++ function), 34
 chimbuko::ADNetClient::connect_ps (C++ function), 34
 chimbuko::ADNetClient::disconnect_ps (C++ function), 34
 chimbuko::ADNetClient::get_client_rank (C++ function), 34
 chimbuko::ADNetClient::get_server_rank (C++ function), 34
 chimbuko::ADNetClient::m_context (C++ member), 34
 chimbuko::ADNetClient::m_rank (C++ member), 34
 chimbuko::ADNetClient::m_socket (C++ member), 34
 chimbuko::ADNetClient::m_srank (C++ member), 34
 chimbuko::ADNetClient::m_use_ps (C++ member), 34
 chimbuko::ADNetClient::send_and_receive (C++ function), 34
 chimbuko::ADNetClient::use_ps (C++ function), 34
 chimbuko::ADOutlier (C++ class), 35
 chimbuko::ADOutlier::~ADOutlier (C++ function), 35
 chimbuko::ADOutlier::ADOutlier (C++ function), 35
 chimbuko::ADOutlier::compute_outliers (C++ function), 36
 chimbuko::ADOutlier::get_global_parameters (C++ function), 35
 chimbuko::ADOutlier::linkExecDataMap (C++ function), 35
 chimbuko::ADOutlier::linkNetworkClient (C++ function), 35
 chimbuko::ADOutlier::linkPerf (C++ function), 35
 chimbuko::ADOutlier::m_execDataMap (C++ member), 36
 chimbuko::ADOutlier::m_local_func_exec_count (C++ member), 36
 chimbuko::ADOutlier::m_net_client (C++ member), 36
 chimbuko::ADOutlier::m_param (C++ member), 36
 chimbuko::ADOutlier::m_perf (C++ member), 36
 chimbuko::ADOutlier::m_rank (C++ member), 36

`chimbuko::ADOutlier::m_use_ps` (C++ member), 36

`chimbuko::ADOutlier::run` (C++ function), 35

`chimbuko::ADOutlier::sync_param` (C++ function), 36

`chimbuko::ADOutlier::use_ps` (C++ function), 35

`chimbuko::ADOutlierSSTD` (C++ class), 36

`chimbuko::ADOutlierSSTD::~ADOutlierSSTD` (C++ function), 36

`chimbuko::ADOutlierSSTD::ADOutlierSSTD` (C++ function), 36

`chimbuko::ADOutlierSSTD::compute_outliers` (C++ function), 37

`chimbuko::ADOutlierSSTD::m_sigma` (C++ member), 37

`chimbuko::ADOutlierSSTD::run` (C++ function), 37

`chimbuko::ADOutlierSSTD::set_sigma` (C++ function), 36

`chimbuko::ADOutlierSSTD::sync_param` (C++ function), 37

`chimbuko::ADParser` (C++ class), 37

`chimbuko::ADParser::~ADParser` (C++ function), 38

`chimbuko::ADParser::addCommData` (C++ function), 40

`chimbuko::ADParser::addCounterData` (C++ function), 40

`chimbuko::ADParser::addFuncData` (C++ function), 39

`chimbuko::ADParser::ADParser` (C++ function), 38

`chimbuko::ADParser::beginStep` (C++ function), 38

`chimbuko::ADParser::createAndValidateEvent` (C++ function), 41

`chimbuko::ADParser::endStep` (C++ function), 38

`chimbuko::ADParser::fetchCommData` (C++ function), 39

`chimbuko::ADParser::fetchCounterData` (C++ function), 39

`chimbuko::ADParser::fetchFuncData` (C++ function), 38

`chimbuko::ADParser::getCommData` (C++ function), 39

`chimbuko::ADParser::getCounterData` (C++ function), 39

`chimbuko::ADParser::getCounterMap` (C++ function), 38

`chimbuko::ADParser::getCurrentStep` (C++ function), 38

`chimbuko::ADParser::getEarliest` (C++ function), 42

`chimbuko::ADParser::getEvents` (C++ function), 39

`chimbuko::ADParser::getEventType` (C++ function), 38

`chimbuko::ADParser::getFuncData` (C++ function), 39

`chimbuko::ADParser::getFuncMap` (C++ function), 38

`chimbuko::ADParser::getGlobalFunctionIndex` (C++ function), 40

`chimbuko::ADParser::getNewMetaData` (C++ function), 39

`chimbuko::ADParser::getNumCommData` (C++ function), 39

`chimbuko::ADParser::getNumCounterData` (C++ function), 39

`chimbuko::ADParser::getNumFuncData` (C++ function), 39

`chimbuko::ADParser::getStatus` (C++ function), 38

`chimbuko::ADParser::linkNetClient` (C++ function), 38

`chimbuko::ADParser::linkPerf` (C++ function), 38

`chimbuko::ADParser::m_ad` (C++ member), 41

`chimbuko::ADParser::m_attr_once` (C++ member), 41

`chimbuko::ADParser::m_comm_count` (C++ member), 41

`chimbuko::ADParser::m_comm_timestamps` (C++ member), 41

`chimbuko::ADParser::m_counter_count` (C++ member), 41

`chimbuko::ADParser::m_counter_timestamps` (C++ member), 42

`chimbuko::ADParser::m_counterMap` (C++ member), 41

`chimbuko::ADParser::m_current_step` (C++ member), 41

`chimbuko::ADParser::m_engineType` (C++ member), 41

`chimbuko::ADParser::m_event_timestamps` (C++ member), 41

`chimbuko::ADParser::m_eventType` (C++ member), 41

`chimbuko::ADParser::m_funcMap` (C++ member), 41

`chimbuko::ADParser::m_global_func_idx_map` (C++ member), 42

`chimbuko::ADParser::m_inputFile` (C++ member), 41

`chimbuko::ADParser::m_io` (C++ member), 41

`chimbuko::ADParser::m_metadata_seen`

(C++ member), 41
 chimbuko::ADParser::m_new_metadata (C++ member), 41
 chimbuko::ADParser::m_opened (C++ member), 41
 chimbuko::ADParser::m_perf (C++ member), 42
 chimbuko::ADParser::m_reader (C++ member), 41
 chimbuko::ADParser::m_status (C++ member), 41
 chimbuko::ADParser::m_timer_event_count (C++ member), 41
 chimbuko::ADParser::setCommDataCapacity (C++ function), 40
 chimbuko::ADParser::setCounterDataCapacity (C++ function), 40
 chimbuko::ADParser::setCounterMap (C++ function), 40
 chimbuko::ADParser::setEventTypeMap (C++ function), 40
 chimbuko::ADParser::setFuncDataCapacity (C++ function), 40
 chimbuko::ADParser::setFuncMap (C++ function), 40
 chimbuko::ADParser::update_attributes (C++ function), 38
 chimbuko::Anomalies (C++ class), 73
 chimbuko::Anomalies::allEvents (C++ function), 74
 chimbuko::Anomalies::funcEvents (C++ function), 74
 chimbuko::Anomalies::insert (C++ function), 74
 chimbuko::Anomalies::m_all_normal_execs (C++ member), 74
 chimbuko::Anomalies::m_all_outliers (C++ member), 74
 chimbuko::Anomalies::m_func_normal_execs (C++ member), 74
 chimbuko::Anomalies::m_func_outliers (C++ member), 74
 chimbuko::Anomalies::nEvents (C++ function), 74
 chimbuko::Anomalies::nFuncEvents (C++ function), 74
 chimbuko::ANOMALY_STATS (C++ enumerator), 68
 chimbuko::AnomalyData (C++ class), 42
 chimbuko::AnomalyData::~AnomalyData (C++ function), 43
 chimbuko::AnomalyData::AnomalyData (C++ function), 43
 chimbuko::AnomalyData::get_app (C++ function), 43
 chimbuko::AnomalyData::get_json (C++ function), 43
 chimbuko::AnomalyData::get_max_ts (C++ function), 43
 chimbuko::AnomalyData::get_min_ts (C++ function), 43
 chimbuko::AnomalyData::get_n_anomalies (C++ function), 43
 chimbuko::AnomalyData::get_rank (C++ function), 43
 chimbuko::AnomalyData::get_stat_id (C++ function), 43
 chimbuko::AnomalyData::get_step (C++ function), 43
 chimbuko::AnomalyData::m_app (C++ member), 43
 chimbuko::AnomalyData::m_max_timestamp (C++ member), 43
 chimbuko::AnomalyData::m_min_timestamp (C++ member), 43
 chimbuko::AnomalyData::m_n_anomalies (C++ member), 43
 chimbuko::AnomalyData::m_rank (C++ member), 43
 chimbuko::AnomalyData::m_stat_id (C++ member), 43
 chimbuko::AnomalyData::m_step (C++ member), 43
 chimbuko::AnomalyData::set (C++ function), 43
 chimbuko::AnomalyStat (C++ class), 43
 chimbuko::AnomalyStat::~AnomalyStat (C++ function), 44
 chimbuko::AnomalyStat::add (C++ function), 44
 chimbuko::AnomalyStat::AnomalyStat (C++ function), 44
 chimbuko::AnomalyStat::get (C++ function), 44
 chimbuko::AnomalyStat::get_data (C++ function), 44
 chimbuko::AnomalyStat::get_n_data (C++ function), 44
 chimbuko::AnomalyStat::get_stats (C++ function), 44
 chimbuko::AnomalyStat::m_data (C++ member), 44
 chimbuko::AnomalyStat::m_mutex (C++ member), 44
 chimbuko::AnomalyStat::m_stats (C++ member), 44
 chimbuko::anyToStr (C++ function), 83
 chimbuko::anyToStr<std::string> (C++

function), 83
chimbuko::ArrayHasher (C++ *class*), 78
chimbuko::ArrayHasher::operator() (C++
function), 78
chimbuko::Back (C++ *enumerator*), 23
chimbuko::Barrier (C++ *class*), 74
chimbuko::Barrier::Barrier (C++ *function*),
74
chimbuko::Barrier::mCond (C++ *member*), 75
chimbuko::Barrier::mCount (C++ *member*), 75
chimbuko::Barrier::mGeneration (C++ *mem-*
ber), 75
chimbuko::Barrier::mMutex (C++ *member*), 75
chimbuko::Barrier::mThreshold (C++ *mem-*
ber), 75
chimbuko::Barrier::wait (C++ *function*), 74
chimbuko::Both (C++ *enumerator*), 23
chimbuko::CallList_t (C++ *type*), 23
chimbuko::CallListIterator_t (C++ *type*), 23
chimbuko::CallStack_t (C++ *type*), 24
chimbuko::CallStackViolation (C++ *enumer-*
ator), 22
chimbuko::Chimbuko (C++ *class*), 15
chimbuko::Chimbuko::~~Chimbuko (C++ *func-*
tion), 15
chimbuko::Chimbuko::Chimbuko (C++ *func-*
tion), 15
chimbuko::Chimbuko::extractCounters
(C++ *function*), 16
chimbuko::Chimbuko::extractEvents (C++
function), 16
chimbuko::Chimbuko::finalize (C++ *func-*
tion), 15
chimbuko::Chimbuko::get_status (C++ *func-*
tion), 15
chimbuko::Chimbuko::get_step (C++ *func-*
tion), 16
chimbuko::Chimbuko::init_counter (C++
function), 16
chimbuko::Chimbuko::init_event (C++ *func-*
tion), 16
chimbuko::Chimbuko::init_io (C++ *function*),
16
chimbuko::Chimbuko::init_metadata_parser
(C++ *function*), 16
chimbuko::Chimbuko::init_net_client
(C++ *function*), 16
chimbuko::Chimbuko::init_outlier (C++
function), 16
chimbuko::Chimbuko::init_parser (C++
function), 16
chimbuko::Chimbuko::initialize (C++ *func-*
tion), 15
chimbuko::Chimbuko::m_counter (C++ *mem-*
ber), 17
chimbuko::Chimbuko::m_event (C++ *member*),
17
chimbuko::Chimbuko::m_io (C++ *member*), 17
chimbuko::Chimbuko::m_is_initialized
(C++ *member*), 17
chimbuko::Chimbuko::m_metadata_parser
(C++ *member*), 17
chimbuko::Chimbuko::m_net_client (C++
member), 17
chimbuko::Chimbuko::m_outlier (C++ *mem-*
ber), 17
chimbuko::Chimbuko::m_params (C++ *mem-*
ber), 17
chimbuko::Chimbuko::m_parser (C++ *mem-*
ber), 17
chimbuko::Chimbuko::m_perf (C++ *member*),
17
chimbuko::Chimbuko::parseInputStep (C++
function), 16
chimbuko::Chimbuko::run (C++ *function*), 16
chimbuko::Chimbuko::show_status (C++
function), 15
chimbuko::Chimbuko::use_ps (C++ *function*),
15
chimbuko::ChimbukoParams (C++ *class*), 17
chimbuko::ChimbukoParams::ChimbukoParams
(C++ *function*), 17
chimbuko::ChimbukoParams::interval_msec
(C++ *member*), 18
chimbuko::ChimbukoParams::only_one_frame
(C++ *member*), 18
chimbuko::ChimbukoParams::outlier_sigma
(C++ *member*), 17
chimbuko::ChimbukoParams::perf_outputpath
(C++ *member*), 18
chimbuko::ChimbukoParams::perf_step
(C++ *member*), 18
chimbuko::ChimbukoParams::print (C++
function), 17
chimbuko::ChimbukoParams::pserver_addr
(C++ *member*), 17
chimbuko::ChimbukoParams::rank (C++ *mem-*
ber), 18
chimbuko::ChimbukoParams::trace_data_dir
(C++ *member*), 17
chimbuko::ChimbukoParams::trace_engineType
(C++ *member*), 17
chimbuko::ChimbukoParams::trace_inputFile
(C++ *member*), 17
chimbuko::ChimbukoParams::verbose (C++
member), 18
chimbuko::ChimbukoParams::viz_addr (C++
member), 18

chimbuko::ChimbukoParams::viz_anom_winSize [member](#)), [46](#)
 (C++ [member](#)), [18](#) chimbuko::CommData_t::m_pid (C++ [member](#)),
 chimbuko::ChimbukoParams::viz_datadump_outputPath [46](#)
 (C++ [member](#)), [18](#) chimbuko::CommData_t::m_rid (C++ [member](#)),
 chimbuko::ChimbukoParams::viz_iomode [46](#)
 (C++ [member](#)), [18](#) chimbuko::CommData_t::m_src (C++ [member](#)),
 chimbuko::CMD (C++ [enumerator](#)), [68](#) [46](#)
 chimbuko::COMM (C++ [enumerator](#)), [23](#) chimbuko::CommData_t::m_tag (C++ [member](#)),
 chimbuko::commandLineArg (C++ [class](#)), [75](#) [46](#)
 chimbuko::commandLineArg::commandLineArgchimbuko::CommData_t::m_tar (C++ [member](#)),
 (C++ [function](#)), [75](#) [46](#)
 chimbuko::commandLineArg::help (C++ [function](#)), [75](#) chimbuko::CommData_t::m_tid (C++ [member](#)),
 (C++ [function](#)), [75](#) [46](#)
 chimbuko::commandLineArg::m_arg (C++ chimbuko::CommData_t::m_ts (C++ [member](#)),
[member](#)), [75](#) [46](#)
 chimbuko::commandLineArg::m_help_str chimbuko::CommData_t::set_exec_key (C++
 (C++ [member](#)), [75](#) [function](#)), [45](#)
 chimbuko::commandLineArg::parse (C++ chimbuko::CommData_t::src (C++ [function](#)), [45](#)
[function](#)), [75](#) chimbuko::CommData_t::tar (C++ [function](#)), [45](#)
 chimbuko::commandLineArgBase (C++ [class](#)), chimbuko::CommData_t::ts (C++ [function](#)), [45](#)
[75](#) chimbuko::CommData_t::type (C++ [function](#)),
 chimbuko::commandLineArgBase::~~commandLineArgBase [45](#)
 (C++ [function](#)), [76](#) chimbuko::CommStack_t (C++ [type](#)), [23](#)
 chimbuko::commandLineArgBase::help (C++ chimbuko::COUNT (C++ [enumerator](#)), [23](#)
[function](#)), [76](#) chimbuko::COUNTER_STATS (C++ [enumerator](#)),
 chimbuko::commandLineArgBase::parse [68](#)
 (C++ [function](#)), [76](#) chimbuko::CounterData_t (C++ [class](#)), [46](#)
 chimbuko::commandLineParser (C++ [class](#)), [76](#) chimbuko::CounterData_t::CounterData_t
 chimbuko::commandLineParser::addArg (C++ [function](#)), [76](#) (C++ [function](#)), [46](#)
 chimbuko::commandLineParser::help (C++ chimbuko::CounterData_t::get_counterid
[function](#)), [76](#) (C++ [function](#)), [47](#)
 chimbuko::commandLineParser::m_args chimbuko::CounterData_t::get_countername
 (C++ [member](#)), [76](#) (C++ [function](#)), [47](#)
 chimbuko::commandLineParser::parse (C++ chimbuko::CounterData_t::get_exec_key
[function](#)), [76](#) (C++ [function](#)), [47](#)
 chimbuko::commandLineParser::StructType chimbuko::CounterData_t::get_json (C++
 (C++ [type](#)), [76](#) [function](#)), [46](#)
 chimbuko::CommData_t (C++ [class](#)), [45](#) chimbuko::CounterData_t::get_pid (C++
 chimbuko::CommData_t::~~CommData_t (C++ [function](#)), [46](#)
[function](#)), [45](#) chimbuko::CounterData_t::get_rid (C++
 chimbuko::CommData_t::CommData_t (C++ [function](#)), [45](#) [function](#)), [47](#)
 chimbuko::CommData_t::get_exec_key (C++ chimbuko::CounterData_t::get_tid (C++
[function](#)), [45](#) [function](#)), [47](#)
 chimbuko::CommData_t::get_json (C++ [function](#)), [46](#) chimbuko::CounterData_t::get_ts (C++
[function](#)), [46](#) [function](#)), [47](#)
 chimbuko::CommData_t::is_same (C++ [function](#)), [45](#) chimbuko::CounterData_t::get_value (C++
 chimbuko::CommData_t::m_bytes (C++ [member](#)), [46](#) [function](#)), [47](#)
 chimbuko::CommData_t::m_commType (C++ chimbuko::CounterData_t::m_cid (C++ [member](#)), [47](#)
[member](#)), [46](#) [function](#)), [47](#)
 chimbuko::CommData_t::m_execkey (C++ chimbuko::CounterData_t::m_countername
[member](#)), [46](#) (C++ [member](#)), [47](#)
 chimbuko::CommData_t::m_execkey (C++ chimbuko::CounterData_t::m_execkey (C++
[member](#)), [46](#) [member](#)), [47](#)
 chimbuko::CommData_t::m_execkey (C++ chimbuko::CounterData_t::m_pid (C++ [member](#)), [47](#)

ber), 47
 chimbuko::CounterData_t::m_rid (C++ *member*), 47
 chimbuko::CounterData_t::m_tid (C++ *member*), 47
 chimbuko::CounterData_t::m_ts (C++ *member*), 47
 chimbuko::CounterData_t::m_value (C++ *member*), 47
 chimbuko::CounterData_t::set_exec_key (C++ *function*), 47
 chimbuko::CounterDataList_t (C++ *type*), 19
 chimbuko::CounterDataListIterator_t (C++ *type*), 19
 chimbuko::CountersByIndex_t (C++ *type*), 19
 chimbuko::CounterStack_t (C++ *type*), 23
 chimbuko::CounterTimeStamps_t (C++ *type*), 19
 chimbuko::DEF_MAP3UL (C++ *function*), 19, 24
 chimbuko::DEFAULT (C++ *enumerator*), 68
 chimbuko::DefaultNetInterface (C++ *type*), 66
 chimbuko::DefaultNetInterface::get (C++ *function*), 66
 chimbuko::DispatchQueue (C++ *class*), 76
 chimbuko::DispatchQueue::~DispatchQueue (C++ *function*), 77
 chimbuko::DispatchQueue::dispatch (C++ *function*), 77
 chimbuko::DispatchQueue::DispatchQueue (C++ *function*), 77
 chimbuko::DispatchQueue::fp_t (C++ *type*), 77
 chimbuko::DispatchQueue::m_cv (C++ *member*), 77
 chimbuko::DispatchQueue::m_lock (C++ *member*), 77
 chimbuko::DispatchQueue::m_name (C++ *member*), 77
 chimbuko::DispatchQueue::m_q (C++ *member*), 77
 chimbuko::DispatchQueue::m_quit (C++ *member*), 77
 chimbuko::DispatchQueue::m_threads (C++ *member*), 77
 chimbuko::DispatchQueue::size (C++ *function*), 77
 chimbuko::DispatchQueue::thread_handler (C++ *function*), 77
 chimbuko::ECHO (C++ *enumerator*), 68
 chimbuko::EmptyCallStack (C++ *enumerator*), 22
 chimbuko::Event_t (C++ *class*), 47
 chimbuko::Event_t::~Event_t (C++ *function*), 48
 chimbuko::Event_t::bytes (C++ *function*), 49
 chimbuko::Event_t::counter_id (C++ *function*), 49
 chimbuko::Event_t::counter_value (C++ *function*), 49
 chimbuko::Event_t::eid (C++ *function*), 48
 chimbuko::Event_t::Event_t (C++ *function*), 48
 chimbuko::Event_t::fid (C++ *function*), 48
 chimbuko::Event_t::get_data_len (C++ *function*), 49
 chimbuko::Event_t::get_json (C++ *function*), 49
 chimbuko::Event_t::get_ptr (C++ *function*), 49
 chimbuko::Event_t::id (C++ *function*), 48
 chimbuko::Event_t::idx (C++ *function*), 48
 chimbuko::Event_t::m_data (C++ *member*), 49
 chimbuko::Event_t::m_id (C++ *member*), 49
 chimbuko::Event_t::m_idx (C++ *member*), 49
 chimbuko::Event_t::m_t (C++ *member*), 49
 chimbuko::Event_t::operator== (C++ *function*), 49
 chimbuko::Event_t::partner (C++ *function*), 48
 chimbuko::Event_t::pid (C++ *function*), 48
 chimbuko::Event_t::rid (C++ *function*), 48
 chimbuko::Event_t::strtype (C++ *function*), 48
 chimbuko::Event_t::tag (C++ *function*), 48
 chimbuko::Event_t::tid (C++ *function*), 48
 chimbuko::Event_t::ts (C++ *function*), 48
 chimbuko::Event_t::type (C++ *function*), 48
 chimbuko::Event_t::valid (C++ *function*), 48
 chimbuko::EventData_t (C++ *enum*), 23
 chimbuko::EventError (C++ *enum*), 22
 chimbuko::EventType (C++ *enum*), 73
 chimbuko::ExecData_t (C++ *class*), 49
 chimbuko::ExecData_t::~ExecData_t (C++ *function*), 50
 chimbuko::ExecData_t::add_counter (C++ *function*), 51
 chimbuko::ExecData_t::add_message (C++ *function*), 51
 chimbuko::ExecData_t::can_delete (C++ *function*), 52
 chimbuko::ExecData_t::ExecData_t (C++ *function*), 50
 chimbuko::ExecData_t::get_counters (C++ *function*), 50
 chimbuko::ExecData_t::get_entry (C++ *function*), 50
 chimbuko::ExecData_t::get_exclusive

(C++ function), 50
 chimbuko::ExecData_t::get_exit (C++ function), 50
 chimbuko::ExecData_t::get_fid (C++ function), 50
 chimbuko::ExecData_t::get_funcname (C++ function), 50
 chimbuko::ExecData_t::get_GPU_correlationID (C++ function), 52
 chimbuko::ExecData_t::get_id (C++ function), 50
 chimbuko::ExecData_t::get_inclusive (C++ function), 50
 chimbuko::ExecData_t::get_json (C++ function), 52
 chimbuko::ExecData_t::get_label (C++ function), 50
 chimbuko::ExecData_t::get_messages (C++ function), 50
 chimbuko::ExecData_t::get_n_children (C++ function), 51
 chimbuko::ExecData_t::get_n_counter (C++ function), 51
 chimbuko::ExecData_t::get_n_message (C++ function), 51
 chimbuko::ExecData_t::get_parent (C++ function), 50
 chimbuko::ExecData_t::get_pid (C++ function), 50
 chimbuko::ExecData_t::get_rid (C++ function), 50
 chimbuko::ExecData_t::get_runtime (C++ function), 50
 chimbuko::ExecData_t::get_tid (C++ function), 50
 chimbuko::ExecData_t::has_GPU_correlationID (C++ function), 52
 chimbuko::ExecData_t::inc_n_children (C++ function), 51
 chimbuko::ExecData_t::is_same (C++ function), 52
 chimbuko::ExecData_t::m_can_delete (C++ member), 53
 chimbuko::ExecData_t::m_counters (C++ member), 53
 chimbuko::ExecData_t::m_entry (C++ member), 52
 chimbuko::ExecData_t::m_exclusive (C++ member), 53
 chimbuko::ExecData_t::m_exit (C++ member), 52
 chimbuko::ExecData_t::m_fid (C++ member), 52
 chimbuko::ExecData_t::m_funcname (C++ member), 52
 chimbuko::ExecData_t::m_gpu_correlation_id_partner (C++ member), 53
 chimbuko::ExecData_t::m_id (C++ member), 52
 chimbuko::ExecData_t::m_label (C++ member), 53
 chimbuko::ExecData_t::m_messages (C++ member), 53
 chimbuko::ExecData_t::m_n_children (C++ member), 53
 chimbuko::ExecData_t::m_n_messages (C++ member), 53
 chimbuko::ExecData_t::m_parent (C++ member), 53
 chimbuko::ExecData_t::m_pid (C++ member), 52
 chimbuko::ExecData_t::m_rid (C++ member), 52
 chimbuko::ExecData_t::m_runtime (C++ member), 52
 chimbuko::ExecData_t::m_tid (C++ member), 52
 chimbuko::ExecData_t::set_funcname (C++ function), 51
 chimbuko::ExecData_t::set_GPU_correlationID_partner (C++ function), 52
 chimbuko::ExecData_t::set_label (C++ function), 51
 chimbuko::ExecData_t::set_parent (C++ function), 51
 chimbuko::ExecData_t::update_exclusive (C++ function), 51
 chimbuko::ExecData_t::update_exit (C++ function), 51
 chimbuko::ExecDataMap_t (C++ type), 24
 chimbuko::Front (C++ enumerator), 23
 chimbuko::FUNC (C++ enumerator), 23
 chimbuko::FUNCTION_INDEX (C++ enumerator), 68
 chimbuko::generate_event_id (C++ function), 54
 chimbuko::generate_hex (C++ function), 54
 chimbuko::GlobalAnomalyStats (C++ class), 58
 chimbuko::GlobalAnomalyStats::~~GlobalAnomalyStats (C++ function), 58
 chimbuko::GlobalAnomalyStats::add_anomaly_data (C++ function), 58
 chimbuko::GlobalAnomalyStats::collect (C++ function), 59
 chimbuko::GlobalAnomalyStats::collect_func_data (C++ function), 59
 chimbuko::GlobalAnomalyStats::collect_stat_data

(C++ function), 59	chimbuko::mapPrint::mapPrint (C++ func-
chimbuko::GlobalAnomalyStats::get_anomaly_stat	tion), 71
(C++ function), 58	chimbuko::mapPrint::mp (C++ member), 71
chimbuko::GlobalAnomalyStats::get_n_anomaly_data	:Message (C++ class), 68
(C++ function), 58	chimbuko::Message::~Message (C++ function),
chimbuko::GlobalAnomalyStats::GlobalAnomalyStat	68
(C++ function), 58	chimbuko::Message::buf (C++ function), 68
chimbuko::GlobalAnomalyStats::m_anomaly_shambuko::Message::clear (C++ function), 69	
(C++ member), 59	chimbuko::Message::createReply (C++ func-
chimbuko::GlobalAnomalyStats::m_exclusive	tion), 69
(C++ member), 59	chimbuko::Message::data (C++ function), 68
chimbuko::GlobalAnomalyStats::m_func	chimbuko::Message::dst (C++ function), 69
(C++ member), 59	chimbuko::Message::frame (C++ function), 69
chimbuko::GlobalAnomalyStats::m_func_anomaly	chimbuko::Message::Header (C++ class), 69
(C++ member), 59	chimbuko::Message::Header::dst (C++ func-
chimbuko::GlobalAnomalyStats::m_inclusive	tion), 69
(C++ member), 59	chimbuko::Message::Header::frame (C++
chimbuko::GlobalAnomalyStats::m_mutex_func	function), 70
(C++ member), 59	chimbuko::Message::Header::get_json
chimbuko::GlobalAnomalyStats::reset_anomaly_stat	(C++ function), 70
(C++ function), 58	chimbuko::Message::Header::Header (C++
chimbuko::GlobalAnomalyStats::update_func_stat	function), 69
(C++ function), 59	chimbuko::Message::Header::kind (C++
chimbuko::GlobalCounterStats (C++ class),	function), 69, 70
60	chimbuko::Message::Header::m_h (C++ mem-
chimbuko::GlobalCounterStats::add_data	ber), 70
(C++ function), 60	chimbuko::Message::Header::set_header
chimbuko::GlobalCounterStats::get_json_state	(C++ function), 70
(C++ function), 60	chimbuko::Message::Header::size (C++
chimbuko::GlobalCounterStats::get_stats	function), 70
(C++ function), 60	chimbuko::Message::Header::src (C++ func-
chimbuko::GlobalCounterStats::m_counter_stats	tion), 69
(C++ member), 61	chimbuko::Message::Header::type (C++
chimbuko::GlobalCounterStats::m_mutex	function), 69
(C++ member), 61	chimbuko::Message::kind (C++ function), 69
chimbuko::GPUVirtualThreadInfo (C++	chimbuko::Message::kind_str (C++ function),
class), 33	69
chimbuko::GPUVirtualThreadInfo::context	chimbuko::Message::m_buf (C++ member), 69
(C++ member), 33	chimbuko::Message::m_head (C++ member), 69
chimbuko::GPUVirtualThreadInfo::device	chimbuko::Message::Message (C++ function),
(C++ member), 33	68
chimbuko::GPUVirtualThreadInfo::get_json	chimbuko::Message::set_info (C++ function),
(C++ function), 33	68
chimbuko::GPUVirtualThreadInfo::GPUVirtualThreadInfo	chimbuko::Message::set_msg (C++ function),
(C++ function), 33	68
chimbuko::GPUVirtualThreadInfo::stream	chimbuko::Message::show (C++ function), 69
(C++ member), 33	chimbuko::Message::size (C++ function), 69
chimbuko::GPUVirtualThreadInfo::thread	chimbuko::Message::src (C++ function), 69
(C++ member), 33	chimbuko::Message::type (C++ function), 69
chimbuko::IOError (C++ enum), 22	chimbuko::MessageCmd (C++ enum), 68
chimbuko::IOMode (C++ enum), 22	chimbuko::MessageKind (C++ enum), 68
chimbuko::IOOpenMode (C++ enum), 23	chimbuko::MessageType (C++ enum), 67
chimbuko::ListEnd (C++ enum), 23	chimbuko::MetaData_t (C++ class), 53
chimbuko::mapPrint (C++ class), 71	chimbuko::MetaData_t::get_comm_rank

(C++ function), 53

chimbuko::MetaData_t::get_descr (C++ function), 53

chimbuko::MetaData_t::get_json (C++ function), 53

chimbuko::MetaData_t::get_tid (C++ function), 53

chimbuko::MetaData_t::get_value (C++ function), 53

chimbuko::MetaData_t::m_descr (C++ member), 54

chimbuko::MetaData_t::m_rank (C++ member), 54

chimbuko::MetaData_t::m_tid (C++ member), 54

chimbuko::MetaData_t::m_value (C++ member), 54

chimbuko::MetaData_t::MetaData_t (C++ function), 53

chimbuko::NetInterface (C++ class), 64

chimbuko::NetInterface::~~NetInterface (C++ function), 64

chimbuko::NetInterface::add_payload (C++ function), 65

chimbuko::NetInterface::finalize (C++ function), 64

chimbuko::NetInterface::init (C++ function), 64

chimbuko::NetInterface::init_thread_pool (C++ function), 65

chimbuko::NetInterface::m_nt (C++ member), 65

chimbuko::NetInterface::m_payloads (C++ member), 65

chimbuko::NetInterface::name (C++ function), 64

chimbuko::NetInterface::NetInterface (C++ function), 64

chimbuko::NetInterface::run (C++ function), 64

chimbuko::NetInterface::stop (C++ function), 64

chimbuko::NetPayloadBase (C++ class), 65

chimbuko::NetPayloadBase::~~NetPayloadBase (C++ function), 65

chimbuko::NetPayloadBase::action (C++ function), 65

chimbuko::NetPayloadBase::check (C++ function), 65

chimbuko::NetPayloadBase::kind (C++ function), 65

chimbuko::NetPayloadBase::type (C++ function), 65

chimbuko::NetPayloadGetParams (C++ class), 54

chimbuko::NetPayloadGetParams::action (C++ function), 54

chimbuko::NetPayloadGetParams::kind (C++ function), 54

chimbuko::NetPayloadGetParams::m_param (C++ member), 55

chimbuko::NetPayloadGetParams::NetPayloadGetParams (C++ function), 54

chimbuko::NetPayloadGetParams::type (C++ function), 54

chimbuko::NetPayloadGlobalFunctionIndexMap (C++ class), 61

chimbuko::NetPayloadGlobalFunctionIndexMap::action (C++ function), 62

chimbuko::NetPayloadGlobalFunctionIndexMap::kind (C++ function), 62

chimbuko::NetPayloadGlobalFunctionIndexMap::m_idxmap (C++ member), 62

chimbuko::NetPayloadGlobalFunctionIndexMap::NetPayloadGlobalFunctionIndexMap (C++ function), 62

chimbuko::NetPayloadGlobalFunctionIndexMap::type (C++ function), 62

chimbuko::NetPayloadHandShake (C++ class), 65

chimbuko::NetPayloadHandShake::action (C++ function), 66

chimbuko::NetPayloadHandShake::kind (C++ function), 66

chimbuko::NetPayloadHandShake::type (C++ function), 66

chimbuko::NetPayloadUpdateAnomalyStats (C++ class), 59

chimbuko::NetPayloadUpdateAnomalyStats::action (C++ function), 60

chimbuko::NetPayloadUpdateAnomalyStats::kind (C++ function), 60

chimbuko::NetPayloadUpdateAnomalyStats::m_global_anomaly_stats (C++ member), 60

chimbuko::NetPayloadUpdateAnomalyStats::NetPayloadUpdateAnomalyStats (C++ function), 60

chimbuko::NetPayloadUpdateAnomalyStats::type (C++ function), 60

chimbuko::NetPayloadUpdateCounterStats (C++ class), 61

chimbuko::NetPayloadUpdateCounterStats::action (C++ function), 61

chimbuko::NetPayloadUpdateCounterStats::kind (C++ function), 61

chimbuko::NetPayloadUpdateCounterStats::m_global_counter_stats (C++ member), 61

chimbuko::NetPayloadUpdateCounterStats::NetPayloadUpdateCounterStats (C++ function), 61

chimbuko::NetPayloadUpdateCounterStats::type (C++ function), 61

(C++ function), 61
 chimbuko::NetPayloadUpdateParams (C++ class), 55
 chimbuko::NetPayloadUpdateParams::action (C++ function), 55
 chimbuko::NetPayloadUpdateParams::kind (C++ function), 55
 chimbuko::NetPayloadUpdateParams::m_param (C++ member), 55
 chimbuko::NetPayloadUpdateParams::NetPayloadUpdateParams (C++ function), 55
 chimbuko::NetPayloadUpdateParams::type (C++ function), 55
 chimbuko::NetThreadLevel (C++ enum), 64
 chimbuko::NoCommData (C++ enumerator), 22
 chimbuko::NoCountData (C++ enumerator), 22
 chimbuko::NoFuncData (C++ enumerator), 22
 chimbuko::Normal (C++ enumerator), 73
 chimbuko::Off (C++ enumerator), 22
 chimbuko::Offline (C++ enumerator), 22
 chimbuko::OK (C++ enumerator), 22
 chimbuko::Online (C++ enumerator), 23
 chimbuko::operator!= (C++ function), 42, 80
 chimbuko::operator+ (C++ function), 80
 chimbuko::operator== (C++ function), 42, 80
 chimbuko::operator> (C++ function), 45
 chimbuko::operator< (C++ function), 45
 chimbuko::operator<< (C++ function), 70
 chimbuko::OutIndexRange (C++ enumerator), 22
 chimbuko::Outlier (C++ enumerator), 73
 chimbuko::PARAMETERS (C++ enumerator), 68
 chimbuko::ParamInterface (C++ class), 55
 chimbuko::ParamInterface::~~ParamInterface (C++ function), 55
 chimbuko::ParamInterface::assign (C++ function), 55
 chimbuko::ParamInterface::clear (C++ function), 55
 chimbuko::ParamInterface::get_function_stats (C++ function), 56
 chimbuko::ParamInterface::m_mutex (C++ member), 56
 chimbuko::ParamInterface::ParamInterface (C++ function), 55
 chimbuko::ParamInterface::serialize (C++ function), 55
 chimbuko::ParamInterface::show (C++ function), 56
 chimbuko::ParamInterface::size (C++ function), 55
 chimbuko::ParamInterface::update (C++ function), 55
 chimbuko::ParserError (C++ enum), 22
 chimbuko::parseVariable (C++ function), 70
 chimbuko::PerfStats (C++ class), 79
 chimbuko::PerfStats::add (C++ function), 79
 chimbuko::PerfStats::PerfStats (C++ function), 79
 chimbuko::PerfStats::setWriteLocation (C++ function), 79
 chimbuko::PerfStats::write (C++ function), 79
 chimbuko::PerfTimer (C++ class), 79
 chimbuko::PerfTimer::elapsed_ms (C++ function), 79
 chimbuko::PerfTimer::elapsed_us (C++ function), 79
 chimbuko::PerfTimer::PerfTimer (C++ function), 79
 chimbuko::PerfTimer::start (C++ function), 79
 chimbuko::PSglobalFunctionIndexMap (C++ class), 62
 chimbuko::PSglobalFunctionIndexMap::lookup (C++ function), 62
 chimbuko::PSglobalFunctionIndexMap::m_fmap (C++ member), 62
 chimbuko::PSglobalFunctionIndexMap::m_mutex (C++ member), 62
 chimbuko::PSstatSender (C++ class), 62
 chimbuko::PSstatSender::~~PSstatSender (C++ function), 62
 chimbuko::PSstatSender::add_payload (C++ function), 63
 chimbuko::PSstatSender::bad (C++ function), 63
 chimbuko::PSstatSender::m_bad (C++ member), 63
 chimbuko::PSstatSender::m_payloads (C++ member), 63
 chimbuko::PSstatSender::m_send_freq (C++ member), 63
 chimbuko::PSstatSender::m_stat_sender (C++ member), 63
 chimbuko::PSstatSender::m_stop_sender (C++ member), 63
 chimbuko::PSstatSender::PSstatSender (C++ function), 62
 chimbuko::PSstatSender::run_stat_sender (C++ function), 63
 chimbuko::PSstatSender::set_send_freq (C++ function), 62
 chimbuko::PSstatSender::stop_stat_sender (C++ function), 63
 chimbuko::PSstatSenderGlobalAnomalyStatsPayload (C++ class), 60
 chimbuko::PSstatSenderGlobalAnomalyStatsPayload::a

(C++ function), 60
 chimbuko::PSstatSenderGlobalAnomalyStatsPhimbako::mRunStats::from_state (C++ function), 81
 chimbuko::PSstatSenderGlobalAnomalyStatsPhimbako::PSstatSenderGlobalAnomalyStatsPayload (C++ member), 60
 chimbuko::PSstatSenderGlobalCounterStatsPhimbako::RunStats::get_json (C++ function), 81
 chimbuko::PSstatSenderGlobalCounterStatsPhimbako::add_stats::get_json_state (C++ function), 81
 chimbuko::PSstatSenderGlobalCounterStatsPhimbako::mRunStats::get_state (C++ function), 81
 chimbuko::PSstatSenderGlobalCounterStatsPhimbako::PSstatSenderGlobalCounterStatsPayload (C++ member), 61
 chimbuko::PSstatSenderGlobalCounterStatsPhimbako::RunStats::kurtosis (C++ function), 81
 chimbuko::PSstatSenderPayloadBase (C++ class), 63
 chimbuko::PSstatSenderPayloadBase::~PSstatSenderPayloadBase::m_do_accumulate (C++ function), 64
 chimbuko::PSstatSenderPayloadBase::add_json (C++ member), 82
 chimbuko::PSstatSenderPayloadBase::do_fetch (C++ function), 63
 chimbuko::PSstatSenderPayloadBase::process_bulkback (C++ function), 81
 chimbuko::PSstatSenderPayloadBase::RunStats::mean (C++ function), 81
 chimbuko::PSstatSenderPayloadBase::RunStats::minimum (C++ function), 81
 chimbuko::QUIT (C++ enumerator), 68
 chimbuko::random_char (C++ function), 54
 chimbuko::Read (C++ enumerator), 23
 chimbuko::REP_ADD (C++ enumerator), 67
 chimbuko::REP_CMD (C++ enumerator), 67
 chimbuko::REP_ECHO (C++ enumerator), 68
 chimbuko::REP_GET (C++ enumerator), 67
 chimbuko::REP_QUIT (C++ enumerator), 67
 chimbuko::REQ_ADD (C++ enumerator), 67
 chimbuko::REQ_CMD (C++ enumerator), 67
 chimbuko::REQ_ECHO (C++ enumerator), 67
 chimbuko::REQ_GET (C++ enumerator), 67
 chimbuko::REQ_QUIT (C++ enumerator), 67
 chimbuko::RunMetric (C++ class), 79
 chimbuko::RunMetric::~RunMetric (C++ function), 80
 chimbuko::RunMetric::add (C++ function), 80
 chimbuko::RunMetric::dump (C++ function), 80
 chimbuko::RunMetric::m_metrics (C++ member), 80
 chimbuko::RunMetric::RunMetric (C++ function), 80
 chimbuko::RunStats (C++ class), 80
 chimbuko::RunStats::~RunStats (C++ function), 81
 chimbuko::RunStats::accumulate (C++ function), 81
 chimbuko::RunStats::clear (C++ function), 81
 chimbuko::RunStats::copy (C++ function), 81
 chimbuko::RunStats::count (C++ function), 81
 chimbuko::RunStats::from_json_state (C++ function), 81
 chimbuko::RunStats::mRunStats::from_state (C++ function), 81
 chimbuko::RunStats::mRunStats::get_json (C++ function), 81
 chimbuko::RunStats::mRunStats::get_state (C++ function), 81
 chimbuko::RunStats::kurtosis (C++ function), 81
 chimbuko::RunStats::m_do_accumulate (C++ member), 82
 chimbuko::RunStats::m_state (C++ member), 82
 chimbuko::RunStats::maximum (C++ function), 81
 chimbuko::RunStats::mean (C++ function), 81
 chimbuko::RunStats::minimum (C++ function), 81
 chimbuko::RunStats::operator+= (C++ function), 81
 chimbuko::RunStats::push (C++ function), 81
 chimbuko::RunStats::RunStats (C++ function), 81
 chimbuko::RunStats::set_do_accumulate (C++ function), 81
 chimbuko::RunStats::set_json_state (C++ function), 81
 chimbuko::RunStats::set_state (C++ function), 81
 chimbuko::RunStats::set_strstate (C++ function), 81
 chimbuko::RunStats::skewness (C++ function), 81
 chimbuko::RunStats::State (C++ class), 82
 chimbuko::RunStats::State (C++ type), 80
 chimbuko::RunStats::State::acc (C++ member), 82
 chimbuko::RunStats::State::clear (C++ function), 82
 chimbuko::RunStats::State::count (C++ member), 82
 chimbuko::RunStats::State::eta (C++ member), 82
 chimbuko::RunStats::State::max (C++ member), 82
 chimbuko::RunStats::State::min (C++ member), 82
 chimbuko::RunStats::State::phi (C++ member), 82

ber), 82
 chimbuko::RunStats::State::rho (C++ member), 82
 chimbuko::RunStats::State::State (C++ function), 82
 chimbuko::RunStats::State::tau (C++ member), 82
 chimbuko::RunStats::stddev (C++ function), 81
 chimbuko::RunStats::variance (C++ function), 81
 chimbuko::SstdParam (C++ class), 56
 chimbuko::SstdParam::~SstdParam (C++ function), 56
 chimbuko::SstdParam::assign (C++ function), 56, 57
 chimbuko::SstdParam::clear (C++ function), 56
 chimbuko::SstdParam::deserialize (C++ function), 57
 chimbuko::SstdParam::get_function_stats (C++ function), 57
 chimbuko::SstdParam::get_runstats (C++ function), 57
 chimbuko::SstdParam::m_runstats (C++ member), 58
 chimbuko::SstdParam::operator[] (C++ function), 57
 chimbuko::SstdParam::serialize (C++ function), 56, 57
 chimbuko::SstdParam::show (C++ function), 57
 chimbuko::SstdParam::size (C++ function), 56
 chimbuko::SstdParam::SstdParam (C++ function), 56
 chimbuko::SstdParam::update (C++ function), 56, 57
 chimbuko::SstdParam::update_and_return (C++ function), 57
 chimbuko::static_mean (C++ function), 80
 chimbuko::static_std (C++ function), 80
 chimbuko::stringize (C++ function), 83
 chimbuko::strToAny (C++ function), 83
 chimbuko::strToAny<std::string> (C++ function), 83
 chimbuko::THREAD_MULTIPLE (C++ enumerator), 64
 chimbuko::Unknown (C++ enumerator), 23
 chimbuko::UnknownEvent (C++ enumerator), 22
 chimbuko::UnknownFunc (C++ enumerator), 22
 chimbuko::varBase (C++ class), 71
 chimbuko::varBase::~~varBase (C++ function), 71
 chimbuko::varBase::get (C++ function), 71
 chimbuko::varBase::name (C++ member), 71
 chimbuko::varBase::put (C++ function), 71
 chimbuko::varBase::value (C++ function), 71
 chimbuko::varBase::varBase (C++ function), 71
 chimbuko::varPOD (C++ class), 71
 chimbuko::varPOD::get (C++ function), 72
 chimbuko::varPOD::put (C++ function), 72
 chimbuko::varPOD::val (C++ member), 72
 chimbuko::varPOD::value (C++ function), 72
 chimbuko::varPOD::varPOD (C++ function), 72
 chimbuko::varTensor (C++ class), 72
 chimbuko::varTensor::get (C++ function), 72
 chimbuko::varTensor::getShape (C++ function), 72
 chimbuko::varTensor::map (C++ function), 73
 chimbuko::varTensor::operator() (C++ function), 72
 chimbuko::varTensor::put (C++ function), 72
 chimbuko::varTensor::shape (C++ member), 73
 chimbuko::varTensor::unmap (C++ function), 73
 chimbuko::varTensor::val (C++ member), 73
 chimbuko::varTensor::value (C++ function), 72
 chimbuko::varTensor::varTensor (C++ function), 72
 chimbuko::vecPrint (C++ class), 73
 chimbuko::vecPrint::mp (C++ member), 73
 chimbuko::vecPrint::vecPrint (C++ function), 73
 chimbuko::Verbose (C++ class), 85
 chimbuko::Verbose::on (C++ function), 86
 chimbuko::Verbose::set_verbose (C++ function), 86
 chimbuko::Verbose::vrb (C++ function), 86
 chimbuko::Write (C++ enumerator), 23
 chimbuko::ZMQNet (C++ class), 66
 chimbuko::ZMQNet::~~ZMQNet (C++ function), 66
 chimbuko::ZMQNet::finalize (C++ function), 66
 chimbuko::ZMQNet::init (C++ function), 66
 chimbuko::ZMQNet::init_thread_pool (C++ function), 67
 chimbuko::ZMQNet::m_context (C++ member), 67
 chimbuko::ZMQNet::m_n_requests (C++ member), 67
 chimbuko::ZMQNet::m_threads (C++ member), 67
 chimbuko::ZMQNet::name (C++ function), 66
 chimbuko::ZMQNet::recv (C++ function), 67
 chimbuko::ZMQNet::recvAndSend (C++ function), 67

chimbuko::ZMQNet::run (C++ *function*), 66
 chimbuko::ZMQNet::send (C++ *function*), 67
 chimbuko::ZMQNet::stop (C++ *function*), 66
 chimbuko::ZMQNet::ZMQNet (C++ *function*), 66
 COMM_EVENT_DIM (C *macro*), 21
 COMM_IDX_BYTES (C *macro*), 21
 COMM_IDX_PARTNER (C *macro*), 21
 COMM_IDX_TAG (C *macro*), 21
 COMM_IDX_TS (C *macro*), 21
 COUNTER_EVENT_DIM (C *macro*), 21
 COUNTER_IDX_ID (C *macro*), 21
 COUNTER_IDX_TS (C *macro*), 21
 COUNTER_IDX_VALUE (C *macro*), 21

D

DEF_MAP3UL (C *macro*), 21
 DefaultThreadPool (C++ *type*), 85
 DefaultThreadPool::getThreadPool (C++ *function*), 85
 DefaultThreadPool::submitJob (C++ *function*), 85

F

FUNC_EVENT_DIM (C *macro*), 21
 FUNC_IDX_F (C *macro*), 21
 FUNC_IDX_TS (C *macro*), 21

I

IDX_E (C *macro*), 21
 IDX_P (C *macro*), 21
 IDX_R (C *macro*), 21
 IDX_T (C *macro*), 21
 IO_VERSION (C *macro*), 21

M

MAX_RUNTIME (C *macro*), 21
 mtQueue (C++ *class*), 78
 mtQueue::~~mtQueue (C++ *function*), 78
 mtQueue::clear (C++ *function*), 78
 mtQueue::empty (C++ *function*), 78
 mtQueue::invalidate (C++ *function*), 78
 mtQueue::is_valid (C++ *function*), 78
 mtQueue::m_cond (C++ *member*), 78
 mtQueue::m_mutex (C++ *member*), 78
 mtQueue::m_queue (C++ *member*), 78
 mtQueue::m_valid (C++ *member*), 78
 mtQueue::mtQueue (C++ *function*), 78
 mtQueue::push (C++ *function*), 78
 mtQueue::size (C++ *function*), 78
 mtQueue::tryPop (C++ *function*), 78
 mtQueue::waitPop (C++ *function*), 78

O

operator!= (C++ *function*), 43, 82

operator+ (C++ *function*), 82
 operator== (C++ *function*), 43, 82
 operator> (C++ *function*), 49
 operator< (C++ *function*), 49

T

threadPool (C++ *class*), 83
 threadPool::~~threadPool (C++ *function*), 83
 threadPool::destroy (C++ *function*), 84
 threadPool::IThreadTask (C++ *class*), 84
 threadPool::IThreadTask::~~IThreadTask (C++ *function*), 84
 threadPool::IThreadTask::execute (C++ *function*), 84
 threadPool::IThreadTask::IThreadTask (C++ *function*), 84
 threadPool::IThreadTask::operator= (C++ *function*), 84
 threadPool::m_done (C++ *member*), 84
 threadPool::m_threads (C++ *member*), 84
 threadPool::m_workQueue (C++ *member*), 84
 threadPool::operator= (C++ *function*), 83
 threadPool::pool_size (C++ *function*), 83
 threadPool::queue_size (C++ *function*), 83
 threadPool::sumit (C++ *function*), 83
 threadPool::TaskFuture (C++ *class*), 84
 threadPool::TaskFuture::~~TaskFuture (C++ *function*), 84
 threadPool::TaskFuture::get (C++ *function*), 84
 threadPool::TaskFuture::m_future (C++ *member*), 85
 threadPool::TaskFuture::operator= (C++ *function*), 84
 threadPool::TaskFuture::TaskFuture (C++ *function*), 84
 threadPool::threadPool (C++ *function*), 83
 threadPool::ThreadTask (C++ *class*), 85
 threadPool::ThreadTask::~~ThreadTask (C++ *function*), 85
 threadPool::ThreadTask::execute (C++ *function*), 85
 threadPool::ThreadTask::m_func (C++ *member*), 85
 threadPool::ThreadTask::operator= (C++ *function*), 85
 threadPool::ThreadTask::ThreadTask (C++ *function*), 85
 threadPool::worker (C++ *function*), 84

V

VERBOSE (C *macro*), 85