

BigHybrid - BitDew-MapReduce Module

Generated by Doxygen 1.7.6.1

Wed Aug 13 2014 18:41:58

Contents

1	Data Structure Index	1
1.1	Data Structures	1
2	File Index	3
2.1	File List	3
3	Data Structure Documentation	5
3.1	config_s Struct Reference	5
3.1.1	Field Documentation	5
3.1.1.1	amount_of_tasks_mra	5
3.1.1.2	grid_average_speed	5
3.1.1.3	grid_cpu_power	5
3.1.1.4	initialized	5
3.1.1.5	mra_chunk_count	5
3.1.1.6	mra_chunk_replicas	6
3.1.1.7	mra_chunk_size	6
3.1.1.8	mra_heartbeat_interval	6
3.1.1.9	mra_number_of_workers	6
3.1.1.10	slots_mra	6
3.1.1.11	workers_mra	6
3.2	job_s Struct Reference	6
3.2.1	Field Documentation	7
3.2.1.1	finished	7
3.2.1.2	map_output	7
3.2.1.3	mra_heartbeats	7
3.2.1.4	task_instances	7

3.2.1.5	task_list	7
3.2.1.6	task_status	7
3.2.1.7	tasks_pending	7
3.3	mra_heartbeat_s Struct Reference	7
3.3.1	Detailed Description	7
3.3.2	Field Documentation	7
3.3.2.1	slots_av	7
3.4	stats_s Struct Reference	8
3.4.1	Field Documentation	8
3.4.1.1	map_local	8
3.4.1.2	map_spec_l	8
3.4.1.3	map_spec_r	8
3.4.1.4	mra_map_remote	8
3.4.1.5	reduce_normal	8
3.4.1.6	reduce_spec	8
3.5	task_info_s Struct Reference	8
3.5.1	Detailed Description	9
3.5.2	Field Documentation	9
3.5.2.1	id	9
3.5.2.2	map_output_copied	9
3.5.2.3	phase	9
3.5.2.4	pid	9
3.5.2.5	shuffle_mra_end	9
3.5.2.6	src	9
3.5.2.7	task	9
3.5.2.8	wid	9
3.6	user_s Struct Reference	9
3.6.1	Field Documentation	10
3.6.1.1	map_mra_output_f	10
3.6.1.2	mra_dfs_f	10
3.6.1.3	task_mra_cost_f	10
3.7	w_info_s Struct Reference	10
3.7.1	Field Documentation	10
3.7.1.1	wid	10

4	File Documentation	11
4.1	examples/hello.c File Reference	11
4.1.1	Function Documentation	12
4.1.1.1	main	12
4.1.1.2	mra_map_mra_output_function	12
4.1.1.3	mra_task_mra_cost_function	13
4.1.1.4	read_mra_config_file	13
4.2	include/common-mra.h File Reference	14
4.2.1	Define Documentation	16
4.2.1.1	DATANODE_MRA_MAILBOX	16
4.2.1.2	MAILBOX_ALIAS_SIZE	16
4.2.1.3	MASTER_MRA_MAILBOX	16
4.2.1.4	MAX_SPECULATIVE_COPIES	16
4.2.1.5	MRA_HEARTBEAT_MIN_INTERVAL	16
4.2.1.6	MRA_HEARTBEAT_TIMEOUT	16
4.2.1.7	NONE	16
4.2.1.8	SMS_FINISH	16
4.2.1.9	SMS_GET_INTER_PAIRS	16
4.2.1.10	SMS_GET_MRA_CHUNK	16
4.2.1.11	SMS_MRA_HEARTBEAT	16
4.2.1.12	SMS_TASK	16
4.2.1.13	SMS_TASK_DONE	16
4.2.1.14	TASK_MRA_MAILBOX	16
4.2.1.15	TASKTRACKER_MRA_MAILBOX	16
4.2.2	Typedef Documentation	17
4.2.2.1	mra_heartbeat_t	17
4.2.2.2	mra_task_info_t	17
4.2.3	Enumeration Type Documentation	17
4.2.3.1	task_status_e	17
4.2.4	Function Documentation	17
4.2.4.1	map_mra_output_size	17
4.2.4.2	maxval	17
4.2.4.3	message_is	18
4.2.4.4	receive	18

4.2.4.5	reduce_mra_input_size	19
4.2.4.6	send	19
4.2.4.7	send_mra_sms	20
4.2.5	Variable Documentation	21
4.2.5.1	avg_task_exec_map	21
4.2.5.2	avg_task_exec_reduce	21
4.2.5.3	config_mra	21
4.2.5.4	dist_bruta	21
4.2.5.5	Fg	21
4.2.5.6	job_mra	21
4.2.5.7	mra_perc	21
4.2.5.8	stats_mra	21
4.2.5.9	task_exec	21
4.2.5.10	user_mra	22
4.3	include/dfs-mra.h File Reference	22
4.3.1	Function Documentation	22
4.3.1.1	data_node_mra	22
4.3.1.2	default_mra_dfs_f	23
4.3.1.3	distribute_data_mra	24
4.3.1.4	find_random_mra_chunk_owner	24
4.3.2	Variable Documentation	25
4.3.2.1	chunk_owner_mra	25
4.4	include/mra.h File Reference	25
4.4.1	Enumeration Type Documentation	26
4.4.1.1	phase_e	26
4.4.2	Function Documentation	26
4.4.2.1	MRA_init	27
4.4.2.2	MRA_main	28
4.4.2.3	MRA_set_dfs_f	28
4.4.2.4	MRA_set_map_mra_output_f	29
4.4.2.5	MRA_set_task_mra_cost_f	29
4.5	include/worker-mra.h File Reference	29
4.5.1	Define Documentation	30
4.5.1.1	MAXIMUM_WORKER_FAILURES	30

4.5.2	Typedef Documentation	30
4.5.2.1	w_mra_info_t	30
4.5.3	Function Documentation	30
4.5.3.1	get_mra_worker_id	30
4.6	src/common-mra.c File Reference	31
4.6.1	Function Documentation	32
4.6.1.1	map_mra_output_size	32
4.6.1.2	maxval	32
4.6.1.3	message_is	33
4.6.1.4	receive	33
4.6.1.5	reduce_mra_input_size	34
4.6.1.6	send	34
4.6.1.7	send_mra_sms	35
4.6.1.8	XBT_LOG_EXTERNAL_DEFAULT_CATEGORY . . .	36
4.7	src/dfs-mra.c File Reference	36
4.7.1	Function Documentation	37
4.7.1.1	data_node_mra	37
4.7.1.2	default_mra_dfs_f	38
4.7.1.3	distribute_data_mra	39
4.7.1.4	find_random_mra_chunk_owner	39
4.7.1.5	send_mra_data	40
4.7.1.6	XBT_LOG_EXTERNAL_DEFAULT_CATEGORY . . .	40
4.8	src/master-mra.c File Reference	40
4.8.1	Function Documentation	42
4.8.1.1	finish_all_mra_task_copies	42
4.8.1.2	is_straggler_mra	42
4.8.1.3	master_mra	43
4.8.1.4	print_mra_config	44
4.8.1.5	print_mra_stats	44
4.8.1.6	send_map_to_mra_worker	44
4.8.1.7	send_mra_task	45
4.8.1.8	send_reduce_to_mra_worker	45
4.8.1.9	set_mra_speculative_tasks	46
4.8.1.10	task_time_elapsed_mra	47

4.8.1.11	XBT_LOG_EXTERNAL_DEFAULT_CATEGORY . . .	47
4.8.2	Variable Documentation	47
4.8.2.1	tasks_log	47
4.9	src/simcore-mra.c File Reference	47
4.9.1	Define Documentation	49
4.9.1.1	MAX_LINE_SIZE	49
4.9.2	Function Documentation	49
4.9.2.1	check_config_mra	49
4.9.2.2	free_mra_global_mem	49
4.9.2.3	init_job_mra	49
4.9.2.4	init_mr_mra_config	50
4.9.2.5	init_mra_config	51
4.9.2.6	init_mra_stats	51
4.9.2.7	master_mra	51
4.9.2.8	MRA_main	53
4.9.2.9	read_mra_config_file	53
4.9.2.10	run_mra_simulation	54
4.9.2.11	worker_mra	55
4.9.2.12	XBT_LOG_NEW_DEFAULT_CATEGORY	56
4.10	src/user-mra.c File Reference	56
4.10.1	Function Documentation	57
4.10.1.1	MRA_init	57
4.10.1.2	MRA_set_dfs_f	58
4.10.1.3	MRA_set_map_mra_output_f	58
4.10.1.4	MRA_set_task_mra_cost_f	58
4.11	src/user.c File Reference	59
4.11.1	Function Documentation	59
4.11.1.1	MRA_init	60
4.11.1.2	MRA_set_dfs_f	60
4.11.1.3	MRA_set_map_mra_output_f	60
4.11.1.4	MRA_set_task_mra_cost_f	60
4.12	src/worker-mra.c File Reference	60
4.12.1	Function Documentation	61
4.12.1.1	compute_mra	61

4.12.1.2	get_mra_chunk	62
4.12.1.3	get_mra_map_output	62
4.12.1.4	get_mra_worker_id	63
4.12.1.5	listen_mra	64
4.12.1.6	mra_heartbeat	65
4.12.1.7	update_mra_map_output	65
4.12.1.8	worker_mra	66
4.12.1.9	XBT_LOG_EXTERNAL_DEFAULT_CATEGORY	67

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

config_s	5
job_s	6
mra_heartbeat_s	
Information sent by the workers with every heartbeat	7
stats_s	8
task_info_s	
Information sent as the task data	8
user_s	9
w_info_s	10

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

examples/ hello.c	11
include/ common-mra.h	14
include/ dfs-mra.h	22
include/ mra.h	25
include/ worker-mra.h	29
src/ common-mra.c	31
src/ dfs-mra.c	36
src/ master-mra.c	40
src/ simcore-mra.c	47
src/ user-mra.c	56
src/ user.c	59
src/ worker-mra.c	60

Chapter 3

Data Structure Documentation

3.1 config_s Struct Reference

```
#include <common-mra.h>
```

Data Fields

- double [mra_chunk_size](#)
- double [grid_average_speed](#)
- double [grid_cpu_power](#)
- int [mra_chunk_count](#)
- int [mra_chunk_replicas](#)
- int [mra_heartbeat_interval](#)
- int [amount_of_tasks_mra](#) [2]
- int [mra_number_of_workers](#)
- int [slots_mra](#) [2]
- int [initialized](#)
- msg_host_t * [workers_mra](#)

3.1.1 Field Documentation

3.1.1.1 int [config_s::amount_of_tasks_mra](#)[2]

3.1.1.2 double [config_s::grid_average_speed](#)

3.1.1.3 double [config_s::grid_cpu_power](#)

3.1.1.4 int [config_s::initialized](#)

3.1.1.5 int [config_s::mra_chunk_count](#)

3.1.1.6 `int config_s::mra_chunk_replicas`

3.1.1.7 `double config_s::mra_chunk_size`

3.1.1.8 `int config_s::mra_heartbeat_interval`

3.1.1.9 `int config_s::mra_number_of_workers`

3.1.1.10 `int config_s::slots_mra[2]`

3.1.1.11 `msg_host_t* config_s::workers_mra`

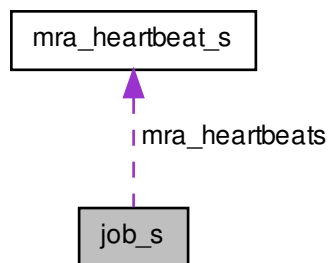
The documentation for this struct was generated from the following file:

- `include/common-mra.h`

3.2 `job_s` Struct Reference

```
#include <common-mra.h>
```

Collaboration diagram for `job_s`:



Data Fields

- `int finished`
- `int tasks_pending [2]`
- `int * task_instances [2]`
- `int * task_status [2]`
- `msg_task_t ** task_list [2]`
- `size_t ** map_output`
- `mra_heartbeat_t mra_heartbeats`

3.2.1 Field Documentation

3.2.1.1 int job_s::finished

3.2.1.2 size_t** job_s::map_output

3.2.1.3 mra_heartbeat_t job_s::mra_heartbeats

3.2.1.4 int* job_s::task_instances[2]

3.2.1.5 msg_task_t** job_s::task_list[2]

3.2.1.6 int* job_s::task_status[2]

3.2.1.7 int job_s::tasks_pending[2]

The documentation for this struct was generated from the following file:

- include/[common-mra.h](#)

3.3 mra_heartbeat_s Struct Reference

Information sent by the workers with every heartbeat.

```
#include <common-mra.h>
```

Data Fields

- int [slots_av](#) [2]

3.3.1 Detailed Description

Information sent by the workers with every heartbeat.

3.3.2 Field Documentation

3.3.2.1 int mra_heartbeat_s::slots_av[2]

The documentation for this struct was generated from the following file:

- include/[common-mra.h](#)

3.4 stats_s Struct Reference

```
#include <common-mra.h>
```

Data Fields

- int [map_local](#)
- int [mra_map_remote](#)
- int [map_spec_l](#)
- int [map_spec_r](#)
- int [reduce_normal](#)
- int [reduce_spec](#)

3.4.1 Field Documentation

3.4.1.1 int stats_s::map_local

3.4.1.2 int stats_s::map_spec_l

3.4.1.3 int stats_s::map_spec_r

3.4.1.4 int stats_s::mra_map_remote

3.4.1.5 int stats_s::reduce_normal

3.4.1.6 int stats_s::reduce_spec

The documentation for this struct was generated from the following file:

- [include/common-mra.h](#)

3.5 task_info_s Struct Reference

Information sent as the task data.

```
#include <common-mra.h>
```

Data Fields

- enum [phase_e](#) [phase](#)
- size_t [id](#)
- size_t [src](#)
- size_t [wid](#)
- int [pid](#)

- `msg_task_t` [task](#)
- `size_t` * [map_output_copied](#)
- `double` [shuffle_mra_end](#)

3.5.1 Detailed Description

Information sent as the task data.

3.5.2 Field Documentation

3.5.2.1 `size_t` `task_info_s::id`

3.5.2.2 `size_t`* `task_info_s::map_output_copied`

3.5.2.3 `enum phase_e` `task_info_s::phase`

3.5.2.4 `int` `task_info_s::pid`

3.5.2.5 `double` `task_info_s::shuffle_mra_end`

3.5.2.6 `size_t` `task_info_s::src`

3.5.2.7 `msg_task_t` `task_info_s::task`

3.5.2.8 `size_t` `task_info_s::wid`

The documentation for this struct was generated from the following file:

- `include/common-mra.h`

3.6 user_s Struct Reference

```
#include <common-mra.h>
```

Data Fields

- `double`(* [task_mra_cost_f](#))(`enum phase_e` phase, `size_t` tid, `size_t` wid)
- `void`(* [mra_dfs_f](#))(`char` **mra_dfs_matrix, `size_t` chunks, `size_t` workers_mra, `int` replicas)
- `int`(* [map_mra_output_f](#))(`size_t` mid, `size_t` rid)

3.6.1 Field Documentation

3.6.1.1 `int(* user_s::map_mra_output_f)(size_t mid, size_t rid)`

3.6.1.2 `void(* user_s::mra_dfs_f)(char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas)`

3.6.1.3 `double(* user_s::task_mra_cost_f)(enum phase_e phase, size_t tid, size_t wid)`

The documentation for this struct was generated from the following file:

- `include/common-mra.h`

3.7 w_info_s Struct Reference

```
#include <worker-mra.h>
```

Data Fields

- `size_t wid`

3.7.1 Field Documentation

3.7.1.1 `size_t w_info_s::wid`

The documentation for this struct was generated from the following file:

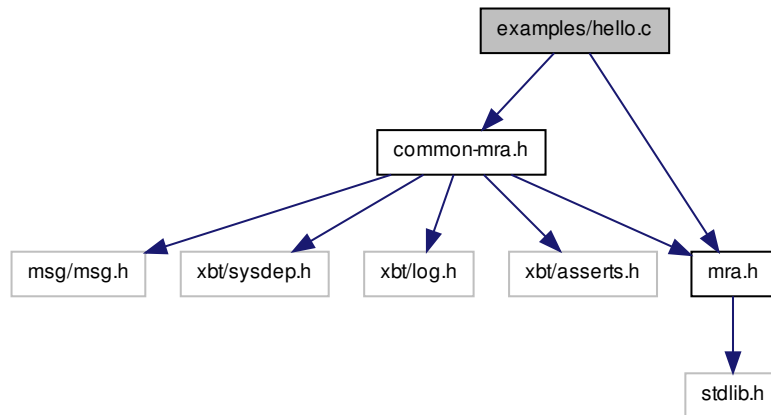
- `include/worker-mra.h`

Chapter 4

File Documentation

4.1 examples/hello.c File Reference

`#include "common-mra.h" #include <mra.h>` Include dependency graph for hello.c:



Functions

- static void [read_mra_config_file](#) (const char *file_name)
- int [mra_map_mra_output_function](#) (size_t mid, size_t rid)
- double [mra_task_mra_cost_function](#) (enum [phase_e](#) phase, size_t tid, size_t wid)
- int [main](#) (int argc, char *argv[])

4.1.1.1 int main (int argc, char * argv[])

The diagram is a complex dependency graph for the 'ray' package. It features a large number of nodes, each representing a module or file, connected by directed edges indicating dependencies. The nodes are organized into several layers, with some nodes at the top (e.g., 'ray_constants', 'ray_constants', 'ray_constants') and others at the bottom (e.g., 'ray_constants', 'ray_constants', 'ray_constants'). The graph shows a high degree of connectivity, with many nodes having multiple incoming and outgoing dependencies. The nodes are labeled with their full paths, such as 'ray_constants', 'ray_constants', 'ray_constants', etc. The edges are represented by blue arrows, showing the direction of the dependency. The overall structure is a dense web of interdependencies, reflecting the complexity of the 'ray' package.

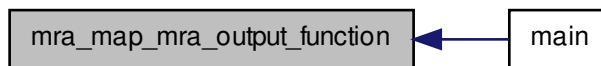
User function that indicates the amount of bytes that a map task will emit to a reduce task.

<i>mid</i>	The ID of the map task.
<i>rid</i>	The ID of the reduce task.

Returns

The amount of data emitted (in bytes).

Here is the caller graph for this function:



4.1.1.3 `double mra_task_mra_cost_function (enum phase_e phase, size_t tid, size_t wid)`

User function that indicates the cost of a task.

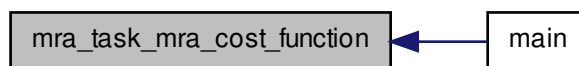
Parameters

<i>phase</i>	The execution phase.
<i>tid</i>	The ID of the task.
<i>wid</i>	The ID of the worker that received the task.

Returns

The task cost in FLOPs.

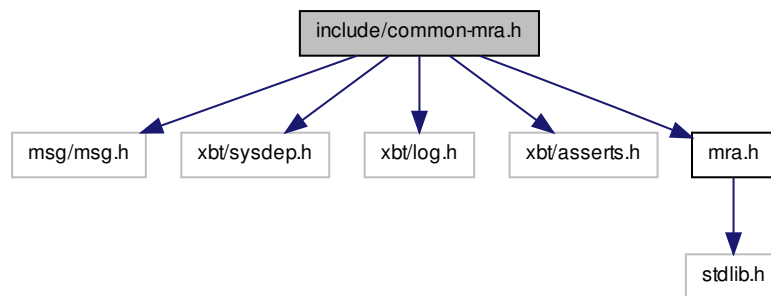
Here is the caller graph for this function:



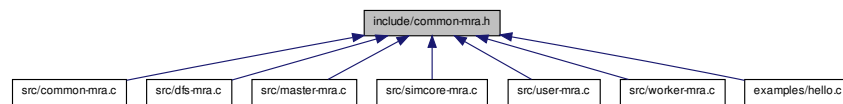
4.1.1.4 `static void read_mra_config_file (const char * file_name)` `[static]`

4.2 include/common-mra.h File Reference

```
#include <msg/msg.h>  #include <xbt/sysdep.h>  #include
<xbt/log.h>  #include <xbt/asserts.h>  #include "mra.h" ×
Include dependency graph for common-mra.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [mra_heartbeat_s](#)
Information sent by the workers with every heartbeat.
- struct [config_s](#)
- struct [job_s](#)
- struct [task_info_s](#)
Information sent as the task data.
- struct [stats_s](#)
- struct [user_s](#)

Defines

- `#define` [MRA_HEARTBEAT_MIN_INTERVAL](#) 3

- #define `MRA_HEARTBEAT_TIMEOUT` 600
- #define `SMS_GET_MRA_CHUNK` "SMS-GC"
- #define `SMS_GET_INTER_PAIRS` "SMS-GIP"
- #define `SMS_MRA_HEARTBEAT` "SMS-HB"
- #define `SMS_TASK` "SMS-T"
- #define `SMS_TASK_DONE` "SMS-TD"
- #define `SMS_FINISH` "SMS-F"
- #define `NONE` (-1)
- #define `MAX_SPECULATIVE_COPIES` 3
- #define `MAILBOX_ALIAS_SIZE` 256
- #define `MASTER_MRA_MAILBOX` "MASTER_MRA"
- #define `DATANODE_MRA_MAILBOX` "%zu:DN"
- #define `TASKTRACKER_MRA_MAILBOX` "%zu:TT"
- #define `TASK_MRA_MAILBOX` "%zu:%d"

Typedefs

- typedef struct `mra_heartbeat_s` * `mra_heartbeat_t`
- typedef struct `task_info_s` * `mra_task_info_t`

Enumerations

- enum `task_status_e` { `T_STATUS_MRA_PENDING`, `T_STATUS_MRA_TIP`, `T_STATUS_MRA_TIP_SLOW`, `T_STATUS_MRA_DONE` }

Possible task status.

Functions

- msg_error_t `send` (const char *str, double cpu, double net, void *data, const char *mailbox)
Send a message/task.
- msg_error_t `send_mra_sms` (const char *str, const char *mailbox)
Send a short message, of size zero.
- msg_error_t `receive` (msg_task_t *msg, const char *mailbox)
Receive a message/task from a mailbox.
- int `message_is` (msg_task_t msg, const char *str)
Compare the message from a task with a string.
- int `maxval` (int a, int b)
Return the maximum of two values.
- size_t `map_mra_output_size` (size_t mid)
Return the output size of a map task.
- size_t `reduce_mra_input_size` (size_t rid)
Return the input size of a reduce task.

Variables

- int * `dist_bruta`
Initialize dist_bruta, task_exec, avg_task_exec.
- double * `task_exec`
- double * `avg_task_exec_map`
- double * `avg_task_exec_reduce`
- int `Fg`
- int `mra_perc`
- struct `config_s config_mra`
- struct `job_s job_mra`
- struct `stats_s stats_mra`
- struct `user_s user_mra`

4.2.1 Define Documentation

- 4.2.1.1 `#define DATANODE_MRA_MAILBOX "%zu:DN"`
- 4.2.1.2 `#define MAILBOX_ALIAS_SIZE 256`
- 4.2.1.3 `#define MASTER_MRA_MAILBOX "MASTER.MRA"`
- 4.2.1.4 `#define MAX_SPECULATIVE_COPIES 3`
- 4.2.1.5 `#define MRA_HEARTBEAT_MIN_INTERVAL 3`
- 4.2.1.6 `#define MRA_HEARTBEAT_TIMEOUT 600`
- 4.2.1.7 `#define NONE (-1)`
- 4.2.1.8 `#define SMS_FINISH "SMS-F"`
- 4.2.1.9 `#define SMS_GET_INTER_PAIRS "SMS-GIP"`
- 4.2.1.10 `#define SMS_GET_MRA_CHUNK "SMS-GC"`
- 4.2.1.11 `#define SMS_MRA_HEARTBEAT "SMS-HB"`
- 4.2.1.12 `#define SMS_TASK "SMS-T"`
- 4.2.1.13 `#define SMS_TASK_DONE "SMS-TD"`
- 4.2.1.14 `#define TASK_MRA_MAILBOX "%zu:%d"`
- 4.2.1.15 `#define TASKTRACKER_MRA_MAILBOX "%zu:TT"`

4.2.2 Typedef Documentation

4.2.2.1 typedef struct mra_heartbeat_s* mra_heartbeat_t

4.2.2.2 typedef struct task_info_s* mra_task_info_t

4.2.3 Enumeration Type Documentation

4.2.3.1 enum task_status_e

Possible task status.

Enumerator:

T_STATUS_MRA_PENDING
T_STATUS_MRA_TIP
T_STATUS_MRA_TIP_SLOW
T_STATUS_MRA_DONE

4.2.4 Function Documentation

4.2.4.1 size_t map_mra_output_size (size_t *mid*)

Return the output size of a map task.

Parameters

<i>mid</i>	The map task ID.
------------	------------------

Returns

The task output size in bytes.

4.2.4.2 int maxval (int *a*, int *b*)

Return the maximum of two values.

Here is the caller graph for this function:



4.2.4.3 int message_is (msg_task_t msg, const char * str)

Compare the message from a task with a string.

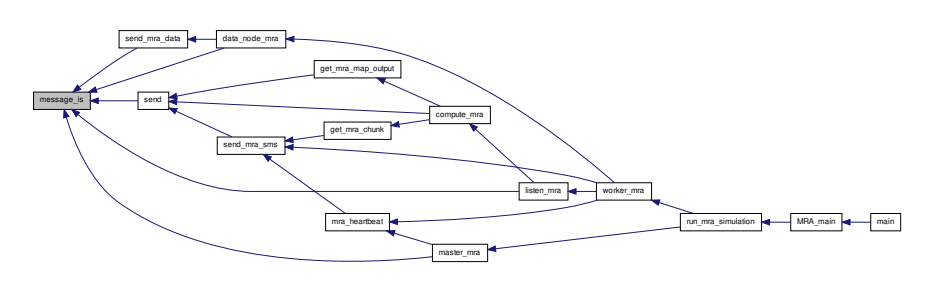
Parameters

msg	The message/task.
str	The string to compare with.

Returns

A positive value if matches, zero if doesn't.

Here is the caller graph for this function:



4.2.4.4 msg_error_t receive (msg_task_t * msg, const char * mailbox)

Receive a message/task from a mailbox.

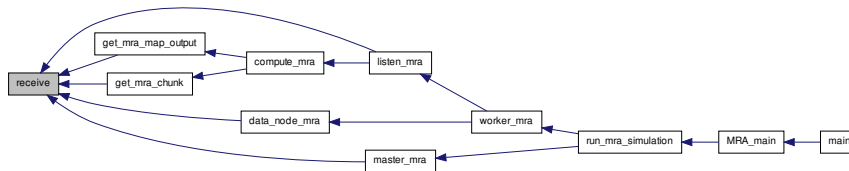
Parameters

msg	Where to store the received message.
mailbox	The mailbox alias.

Returns

The status of the transfer.

Here is the caller graph for this function:

**4.2.4.5 size_t reduce_mra_input_size (size_t rid)**

Return the input size of a reduce task.

Parameters

<i>rid</i>	The reduce task ID.
------------	---------------------

Returns

The task input size in bytes.

Here is the caller graph for this function:

**4.2.4.6 msg_error_t send (const char * str, double cpu, double net, void * data, const char * mailbox)**

Send a message/task.

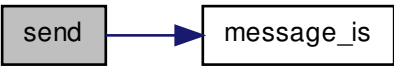
Parameters

<i>str</i>	The message.
<i>cpu</i>	The amount of cpu required by the task.
<i>net</i>	The message size in bytes.
<i>data</i>	Any data to attach to the message.
<i>mailbox</i>	The destination mailbox alias.

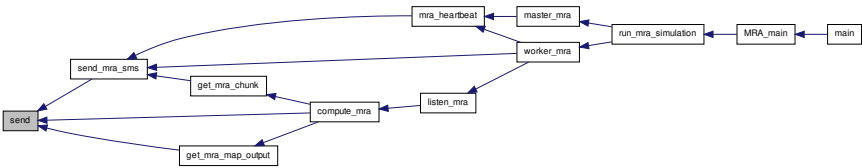
Returns

The MSG status of the operation.

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.4.7 msg_error_t send_mra_sms (const char * str, const char * mailbox)

Send a short message, of size zero.

Parameters

str	The message.
mailbox	The destination mailbox alias.

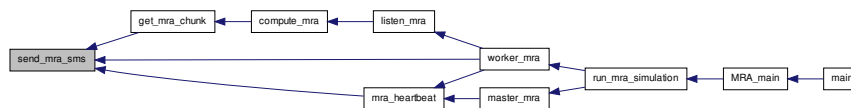
Returns

The MSG status of the operation.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.2.5 Variable Documentation**

4.2.5.1 `double* avg_task_exec_map`

4.2.5.2 `double* avg_task_exec_reduce`

4.2.5.3 `struct config_s config_mra`

4.2.5.4 `int* dist_bruta`

Initialize `dist_bruta`, `task_exec`, `avg_task_exec`.

4.2.5.5 `int Fg`

4.2.5.6 `struct job_s job_mra`

4.2.5.7 `int mra_perc`

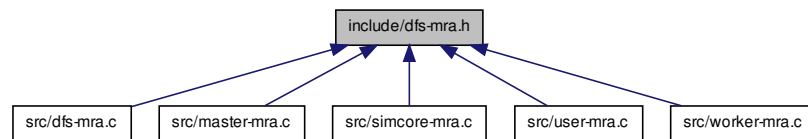
4.2.5.8 `struct stats_s stats_mra`

4.2.5.9 `double* task_exec`

4.2.5.10 struct user_s user_mra

4.3 include/dfs-mra.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void [distribute_data_mra](#) (void)
Distribute chunks (and replicas) to DataNodes.
- void [default_mra_dfs_f](#) (char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas)
Default data distribution algorithm.
- size_t [find_random_mra_chunk_owner](#) (int cid)
Choose a random DataNode that owns a specific chunk.
- int [data_node_mra](#) (int argc, char *argv[])
DataNode main function.

Variables

- char ** [chunk_owner_mra](#)
Matrix that maps chunks to workers.

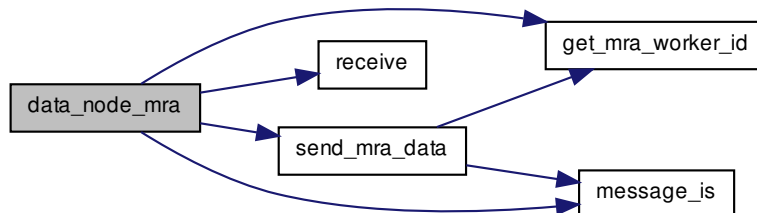
4.3.1 Function Documentation

4.3.1.1 int data_node_mra (int argc, char * argv[])

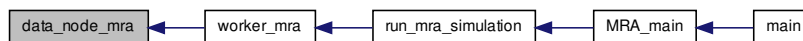
DataNode main function.

Process that listens for data requests.

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.2 void default_mra_dfs_f (char ** mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas)

Default data distribution algorithm.

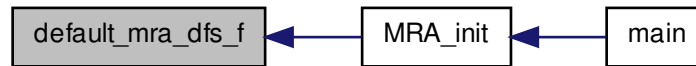
de workers --> workers_hosts[id] (array) capacidade --> MSG_get_host_speed (config_mra.workers[owner]) --> Calcula a capacidade computacional relativa de cada worker baseado na capacidade total da grid. --> É o array com as tribuições brutas, antes do ajuste de menor te_exec --> É o array com o valor de previsão de término de todas as tarefas distribuídas ao worker; --> É o array com o tempo que será utilizado para encontrar a melhor distribuição --> É o array que contém o tempo de cada worker para executar uma tarefa computacional padrão

config_mra.slots_mra[MRA_MAP];

--> verifica qual é o maior tempo de execução previsto

Ajuste de Força Bruta com uma Otimização Combinatória para obter uma distribuição de chunks com o menor tempo de execução possível

Here is the caller graph for this function:



4.3.1.3 void distribute_data_mra (void)

Distribute chunks (and replicas) to DataNodes.

Here is the caller graph for this function:



4.3.1.4 size_t find_random_mra_chunk_owner (int cid)

Choose a random DataNode that owns a specific chunk.

Parameters

<i>cid</i>	The chunk ID.
------------	---------------

Returns

The ID of the DataNode.

Distribution of Data Replication

Parameters

<i>cid</i>	The chunk ID.
------------	---------------

Returns

The ID of the DataNode.

Here is the caller graph for this function:



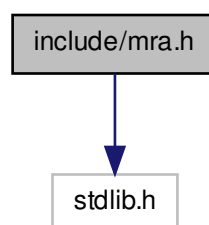
4.3.2 Variable Documentation

4.3.2.1 `char** chunk_owner_mra`

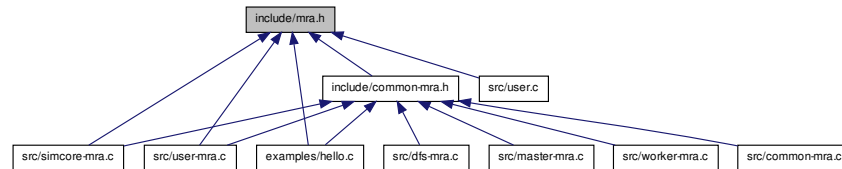
Matrix that maps chunks to workers.

4.4 include/mra.h File Reference

`#include <stdlib.h>` Include dependency graph for mra.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum `phase_e` { `MRA_MAP`, `MRA_REDUCE` }

Possible execution phases.

Functions

- void `MRA_init` (void)
- int `MRA_main` (const char *plat, const char *depl, const char *conf)
- void `MRA_set_task_mra_cost_f` (double(*f)(enum `phase_e` phase, size_t tid, size_t wid))
- void `MRA_set_dfs_f` (void(*f)(char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas))
- void `MRA_set_map_mra_output_f` (int(*f)(size_t mid, size_t rid))

4.4.1 Enumeration Type Documentation

4.4.1.1 enum `phase_e`

Possible execution phases.

Enumerator:

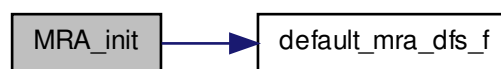
`MRA_MAP`

`MRA_REDUCE`

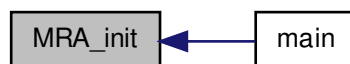
4.4.2 Function Documentation

4.4.2.1 void MRA_init (void)

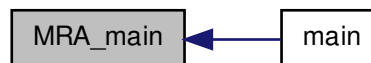
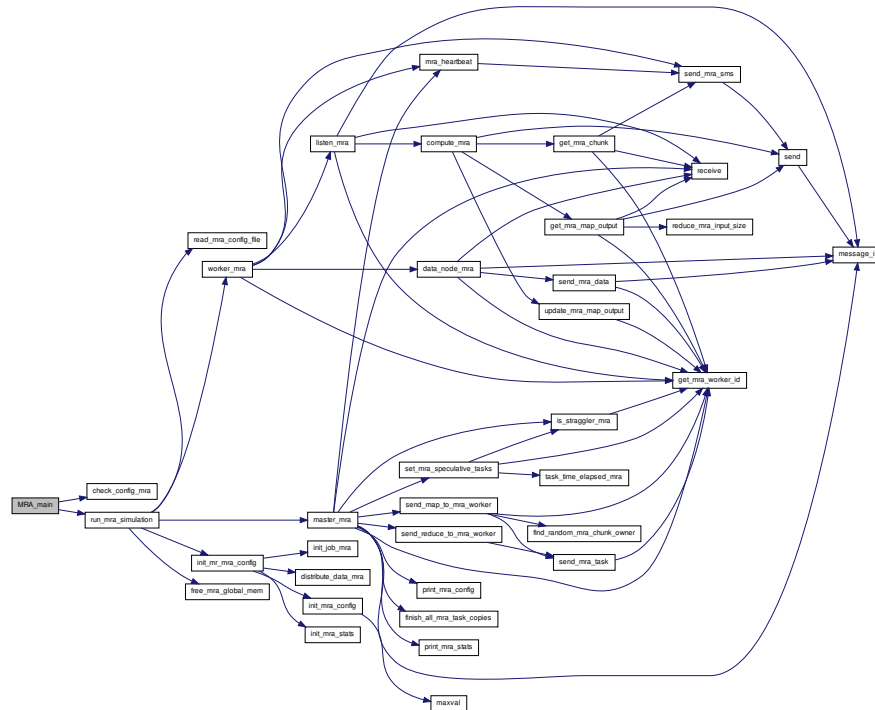
Here is the call graph for this function:



Here is the caller graph for this function:



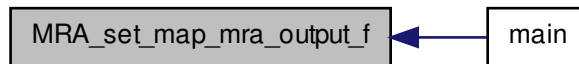
Here is the call graph for this function:



Generated on Wed Aug 13 2014 18:41:58 for BigHybrid - BitDew-MapReduce Module by Doxygen

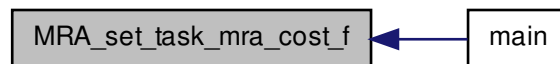
4.4.2.4 void MRA_set_map_mra_output_f (int(*) (size_t mid, size_t rid) f)

Here is the caller graph for this function:



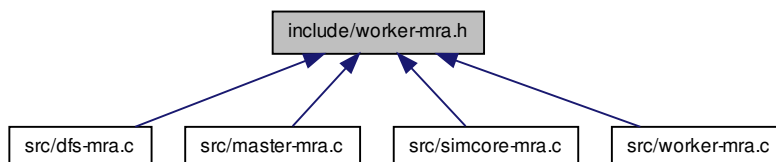
4.4.2.5 void MRA_set_task_mra_cost_f (double(*) (enum phase_e phase, size_t tid, size_t wid) f)

Here is the caller graph for this function:



4.5 include/worker-mra.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [w_info_s](#)

Defines

- #define [MAXIMUM_WORKER_FAILURES](#) 4

Typedefs

- typedef struct [w_info_s](#) * [w_mra_info_t](#)

Functions

- size_t [get_mra_worker_id](#) (msg_host_t worker)
Get the ID of a worker.

4.5.1 Define Documentation

4.5.1.1 #define [MAXIMUM_WORKER_FAILURES](#) 4

4.5.2 Typedef Documentation

4.5.2.1 typedef struct [w_info_s](#)* [w_mra_info_t](#)

4.5.3 Function Documentation

4.5.3.1 size_t [get_mra_worker_id](#) (msg_host_t *worker*)

Get the ID of a worker.

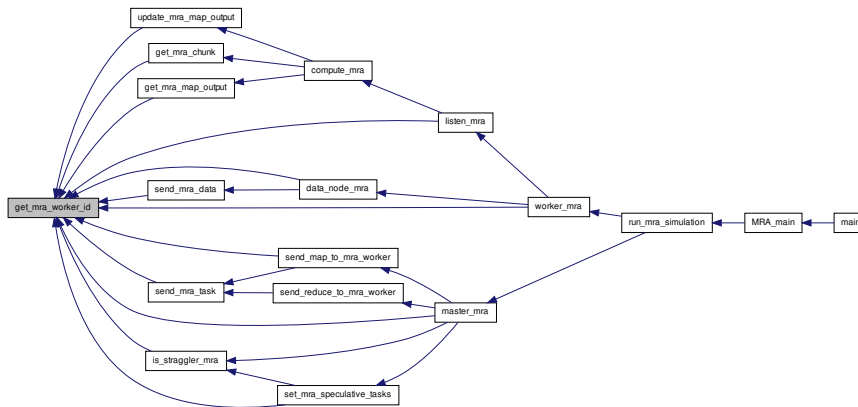
Parameters

<i>worker</i>	The worker node.
---------------	------------------

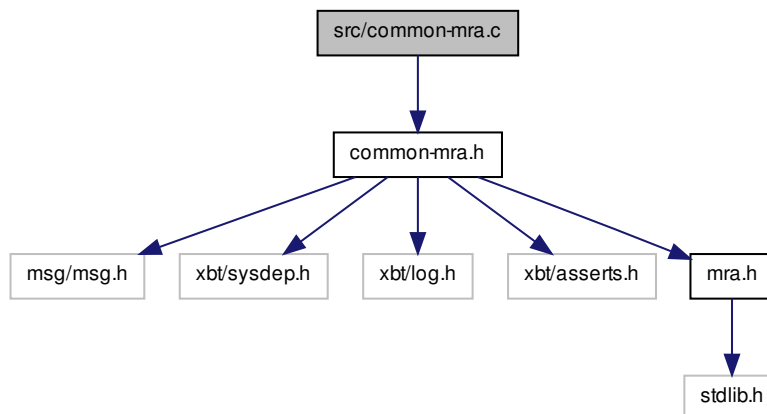
Returns

The worker's ID number.

Here is the caller graph for this function:

**4.6 src/common-mra.c File Reference**

`#include "common-mra.h"` Include dependency graph for `common-mra.c`:



Functions

- [XBT_LOG_EXTERNAL_DEFAULT_CATEGORY](#) (msg_test)
- msg_error_t [send](#) (const char *str, double cpu, double net, void *data, const char *mailbox)
Send a message/task.
- msg_error_t [send_mra_sms](#) (const char *str, const char *mailbox)
Send a short message, of size zero.
- msg_error_t [receive](#) (msg_task_t *msg, const char *mailbox)
Receive a message/task from a mailbox.
- int [message_is](#) (msg_task_t msg, const char *str)
Compare the message from a task with a string.
- int [maxval](#) (int a, int b)
Return the maximum of two values.
- size_t [map_mra_output_size](#) (size_t mid)
Return the output size of a map task.
- size_t [reduce_mra_input_size](#) (size_t rid)
Return the input size of a reduce task.

4.6.1 Function Documentation

4.6.1.1 size_t map_mra_output_size (size_t mid)

Return the output size of a map task.

Parameters

<i>mid</i>	The map task ID.
------------	------------------

Returns

The task output size in bytes.

4.6.1.2 int maxval (int a, int b)

Return the maximum of two values.

Here is the caller graph for this function:



4.6.1.3 int message_is (msg_task_t msg, const char * str)

Compare the message from a task with a string.

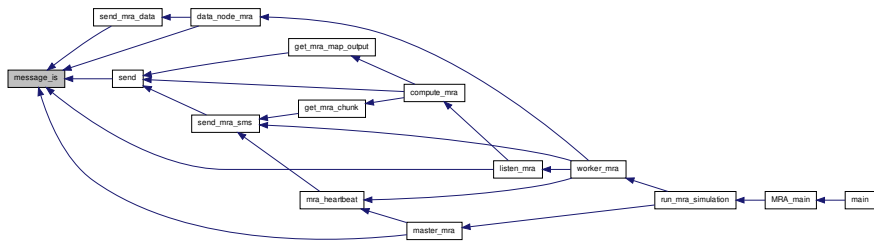
Parameters

<i>msg</i>	The message/task.
<i>str</i>	The string to compare with.

Returns

A positive value if matches, zero if doesn't.

Here is the caller graph for this function:



4.6.1.4 msg_error_t receive (msg_task_t * msg, const char * mailbox)

Receive a message/task from a mailbox.

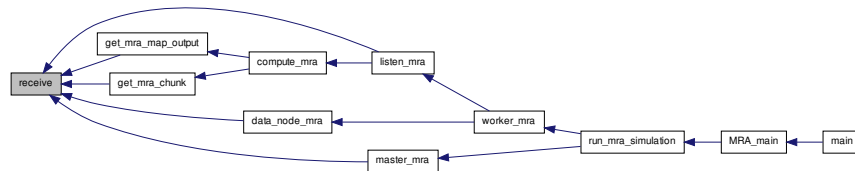
Parameters

<i>msg</i>	Where to store the received message.
<i>mailbox</i>	The mailbox alias.

Returns

The status of the transfer.

Here is the caller graph for this function:

**4.6.1.5 size_t reduce_mra_input_size (size_t rid)**

Return the input size of a reduce task.

Parameters

<i>rid</i>	The reduce task ID.
------------	---------------------

Returns

The task input size in bytes.

Here is the caller graph for this function:

**4.6.1.6 msg_error_t send (const char * str, double cpu, double net, void * data, const char * mailbox)**

Send a message/task.

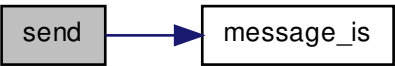
Parameters

<i>str</i>	The message.
<i>cpu</i>	The amount of cpu required by the task.
<i>net</i>	The message size in bytes.
<i>data</i>	Any data to attach to the message.
<i>mailbox</i>	The destination mailbox alias.

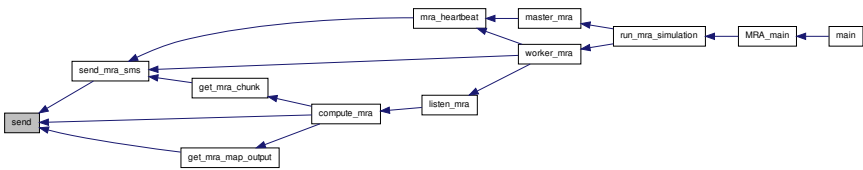
Returns

The MSG status of the operation.

Here is the call graph for this function:



Here is the caller graph for this function:



4.6.1.7 msg_error_t send_mra_sms (const char * str, const char * mailbox)

Send a short message, of size zero.

Parameters

<i>str</i>	The message.
<i>mailbox</i>	The destination mailbox alias.

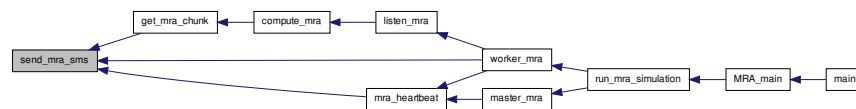
Returns

The MSG status of the operation.

Here is the call graph for this function:



Here is the caller graph for this function:

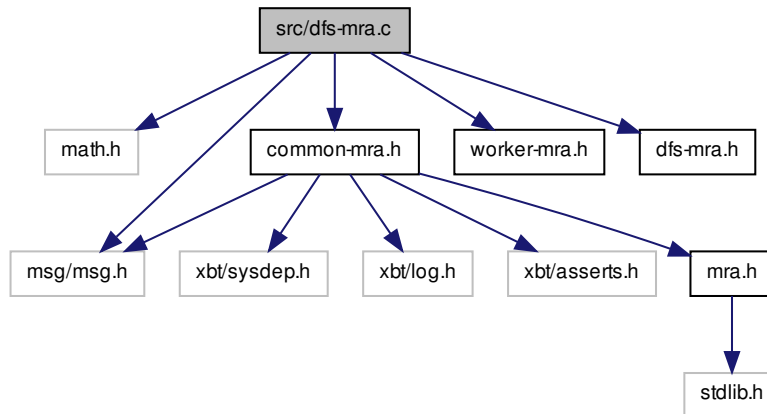


4.6.1.8 XBT_LOG_EXTERNAL_DEFAULT_CATEGORY (msg_test)

4.7 src/dfs-mra.c File Reference

```
#include <math.h> #include <msg/msg.h> #include "common-mra.-
h" #include "worker-mra.h" #include "dfs-mra.h" Include depen-
```

dency graph for dfs-mra.c:



Functions

- `XBT_LOG_EXTERNAL_DEFAULT_CATEGORY` (`msg_test`)
- static void `send_mra_data` (`msg_task_t msg`)
Process that responds to data requests.
- void `distribute_data_mra` (void)
Distribute chunks (and replicas) to DataNodes.
- void `default_mra_dfs_f` (`char **mra_dfs_matrix`, `size_t chunks`, `size_t workers_mra`, `int replicas`)
Default data distribution algorithm.
- `size_t find_random_mra_chunk_owner` (`int cid`)
Choose a random DataNode that owns a specific chunk.
- int `data_node_mra` (`int argc`, `char *argv[]`)
DataNode main function.

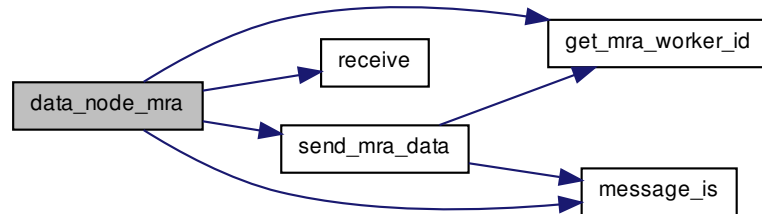
4.7.1 Function Documentation

4.7.1.1 int data_node_mra (int argc, char * argv[])

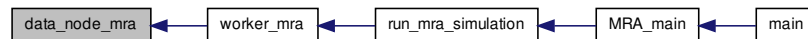
DataNode main function.

Process that listens for data requests.

Here is the call graph for this function:



Here is the caller graph for this function:



4.7.1.2 void default_mra_dfs_f(char ** mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas)

Default data distribution algorithm.

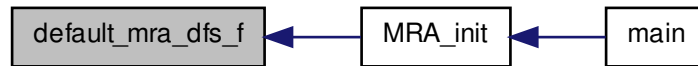
de workers --> workers_hosts[id] (array) capacidade --> MSG_get_host_speed (config_mra.workers[owner]) --> Calcula a capacidade computacional relativa de cada worker baseado na capacidade total da grid. --> É o array com as tribuições brutas, antes do ajuste de menor te_exec --> É o array com o valor de previsão de término de todas as tarefas distribuídas ao worker; --> É o array com o tempo que será utilizado para encontrar a melhor distribuição --> É o array que contém o tempo de cada worker para executar uma tarefa computacional padrão

config_mra.slots_mra[MRA_MAP];

--> verifica qual é o maior tempo de execução previsto

Ajuste de Força Bruta com uma Otimização Combinatória para obter uma distribuição de chunks com o menor tempo de execução possível

Here is the caller graph for this function:



4.7.1.3 void distribute_data_mra (void)

Distribute chunks (and replicas) to DataNodes.

Here is the caller graph for this function:



4.7.1.4 size_t find_random_mra_chunk_owner (int cid)

Choose a random DataNode that owns a specific chunk.

Distribution of Data Replication

Parameters

<i>cid</i>	The chunk ID.
------------	---------------

Returns

The ID of the DataNode.

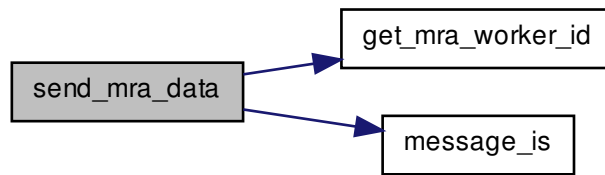
Here is the caller graph for this function:



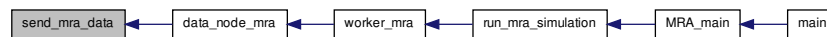
4.7.1.5 static void send_mra_data (msg_task_t msg) [static]

Process that responds to data requests.

Here is the call graph for this function:



Here is the caller graph for this function:

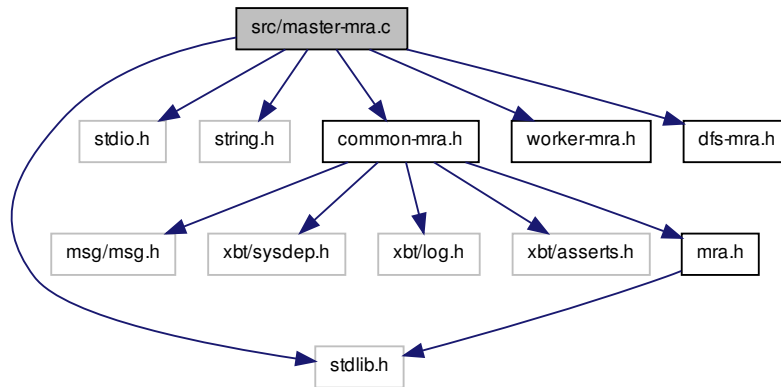


4.7.1.6 XBT_LOG_EXTERNAL_DEFAULT_CATEGORY (msg_test)

4.8 src/master-mra.c File Reference

```
#include <stdlib.h> #include <stdio.h> #include <string.-
h> #include "common-mra.h" #include "worker-mra.h" #include
```

"dfs-mra.h" Include dependency graph for master-mra.c:



Functions

- [XBT_LOG_EXTERNAL_DEFAULT_CATEGORY](#) (msg_test)
- static void [print_mra_config](#) (void)
Print the job configuration.
- static void [print_mra_stats](#) (void)
Print job statistics.
- static int [is_straggler_mra](#) (enum [phase_e](#) phase, msg_host_t worker)
Checks if a worker is a straggler.
- static int [task_time_elapsed_mra](#) (msg_task_t task)
Returns for how long a task is running.
- static void [set_mra_speculative_tasks](#) (enum [phase_e](#) phase, msg_host_t worker)
Mark the tasks of a straggler as possible speculative tasks.
- static void [send_map_to_mra_worker](#) (msg_host_t dest)
Choose a map task, and send it to a worker.
- static void [send_reduce_to_mra_worker](#) (msg_host_t dest)
Choose a reduce task, and send it to a worker.
- static void [send_mra_task](#) (enum [phase_e](#) phase, size_t tid, size_t data_src, msg_host_t dest)
Send a task to a worker.
- static void [finish_all_mra_task_copies](#) (mra_task_info_t ti)
Kill all copies of a task.
- int [master_mra](#) (int argc, char *argv[])
Main master function.

Variables

- static FILE * [tasks_log](#)

4.8.1 Function Documentation

4.8.1.1 static void finish_all_mra_task_copies (mra_task_info_t *ti*) [static]

Kill all copies of a task.

Parameters

<i>ti</i>	The task information of any task instance.
-----------	--

Here is the caller graph for this function:



4.8.1.2 static int is_straggler_mra (enum phase_e *phase*, msg_host_t *worker*) [static]

Checks if a worker is a straggler.

Parameters

<i>worker</i>	The worker to be probed.
---------------	--------------------------

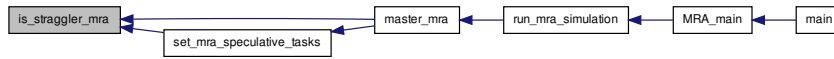
Returns

1 if true, 0 if false.

Here is the call graph for this function:



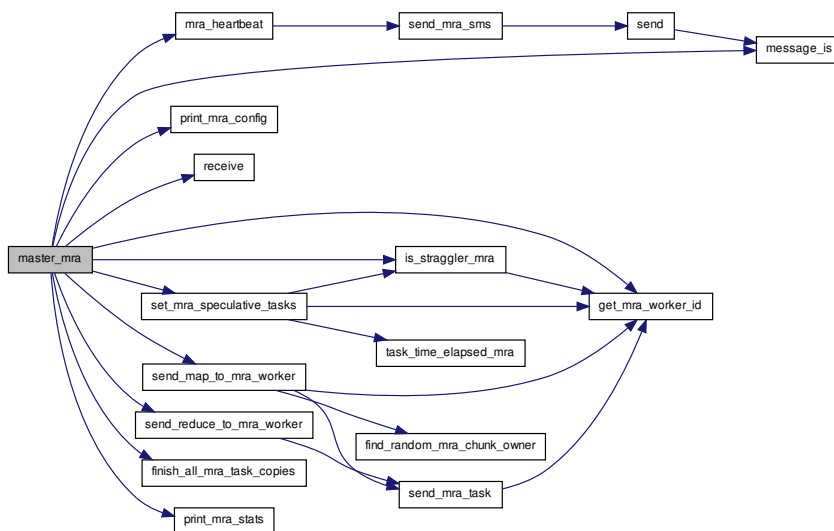
Here is the caller graph for this function:



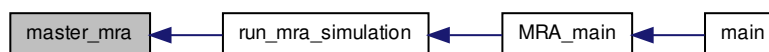
4.8.1.3 int master_mra (int argc, char * argv[])

Main master function.

Here is the call graph for this function:



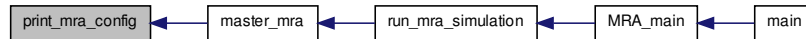
Here is the caller graph for this function:



4.8.1.4 `static void print_mra_config (void) [static]`

Print the job configuration.

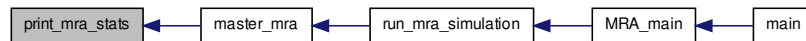
Here is the caller graph for this function:



4.8.1.5 `static void print_mra_stats (void) [static]`

Print job statistics.

Here is the caller graph for this function:



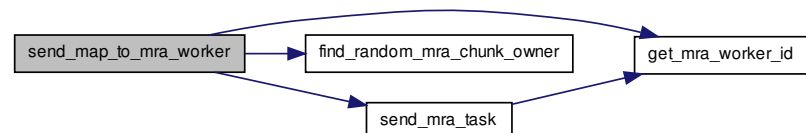
4.8.1.6 `static void send_map_to_mra_worker (msg_host.t dest) [static]`

Choose a map task, and send it to a worker.

Parameters

<i>dest</i>	The destination worker.
-------------	-------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.1.7 static void **send_mra_task** (enum phase_e *phase*, size_t *tid*, size_t *data_src*, msg_host_t *dest*) [static]

Send a task to a worker.

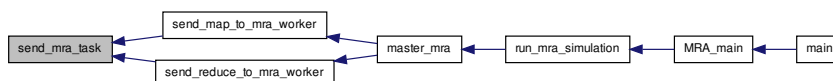
Parameters

<i>phase</i>	The current job phase.
<i>tid</i>	The task ID.
<i>data_src</i>	The ID of the DataNode that owns the task data.
<i>dest</i>	The destination worker.

Here is the call graph for this function:



Here is the caller graph for this function:



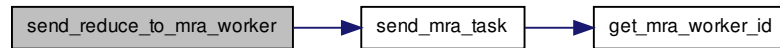
4.8.1.8 static void **send_reduce_to_mra_worker** (msg_host_t *dest*) [static]

Choose a reduce task, and send it to a worker.

Parameters

<i>dest</i>	The destination worker.
-------------	-------------------------

Here is the call graph for this function:



Here is the caller graph for this function:



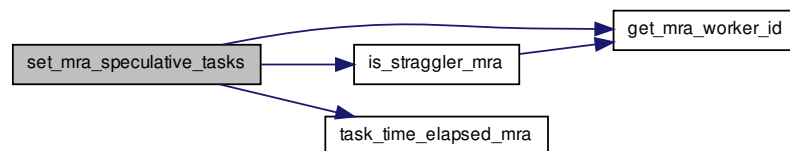
4.8.1.9 static void `set_mra_speculative_tasks` (`enum phase_e phase`, `msg_host_t worker`) [static]

Mark the tasks of a straggler as possible speculative tasks.

Parameters

<i>worker</i>	The straggler worker.
---------------	-----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



4.8.1.10 static int task_time_elapsed_mra (msg_task.t task) [static]

Returns for how long a task is running.

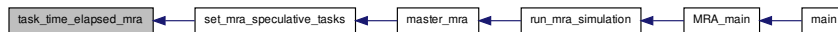
Parameters

<i>task</i>	The task to be probed.
-------------	------------------------

Returns

The amount of seconds since the beginning of the computation.

Here is the caller graph for this function:



4.8.1.11 XBT_LOG_EXTERNAL_DEFAULT_CATEGORY (msg_test)

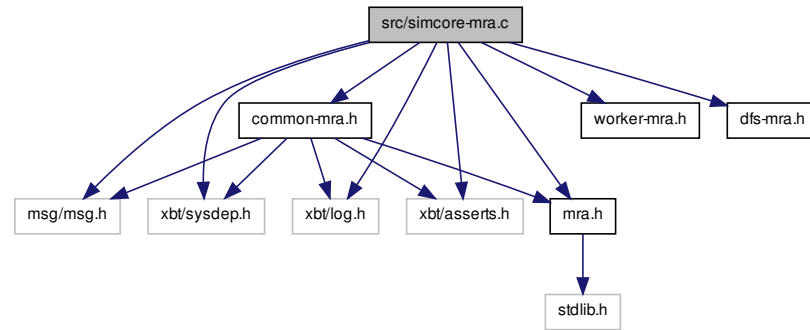
4.8.2 Variable Documentation

4.8.2.1 FILE* tasks_log [static]

4.9 src/simcore-mra.c File Reference

```
#include <msg/msg.h>    #include <xbt/sysdep.h>    #include
<xbt/log.h> #include <xbt/asserts.h> #include "common-mra.-
h" #include "worker-mra.h" #include "dfs-mra.h" #include
```

"mra.h" Include dependency graph for simcore-mra.c:



Defines

- `#define` [MAX_LINE_SIZE](#) 256

Functions

- [XBT_LOG_NEW_DEFAULT_CATEGORY](#) (msg_test,"MRA")
- [int master_mra](#) (int argc, char *argv[])
Main master function.
- [int worker_mra](#) (int argc, char *argv[])
Main worker function.
- [static void check_config_mra](#) (void)
Check if the user configuration is sound.
- [static msg_error_t run_mra_simulation](#) (const char *platform_file, const char *deploy_file, const char *mra_config_file)
- [static void init_mr_mra_config](#) (const char *mra_config_file)
Initialize the MapReduce configuration.
- [static void read_mra_config_file](#) (const char *file_name)
Read the MapReduce configuration file.
- [static void init_mra_config](#) (void)
Initialize the config structure.
- [static void init_job_mra](#) (void)
Initialize the job structure.
- [static void init_mra_stats](#) (void)
Initialize the stats structure.
- [static void free_mra_global_mem](#) (void)
Free allocated memory for global variables.
- [int MRA_main](#) (const char *plat, const char *depl, const char *conf)

4.9.1 Define Documentation

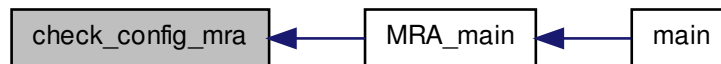
4.9.1.1 `#define MAX_LINE_SIZE 256`

4.9.2 Function Documentation

4.9.2.1 `static void check_config_mra (void) [static]`

Check if the user configuration is sound.

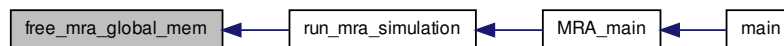
Here is the caller graph for this function:



4.9.2.2 `static void free_mra_global_mem (void) [static]`

Free allocated memory for global variables.

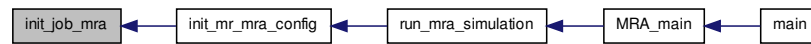
Here is the caller graph for this function:



4.9.2.3 `static void init_job_mra (void) [static]`

Initialize the job structure.

Here is the caller graph for this function:



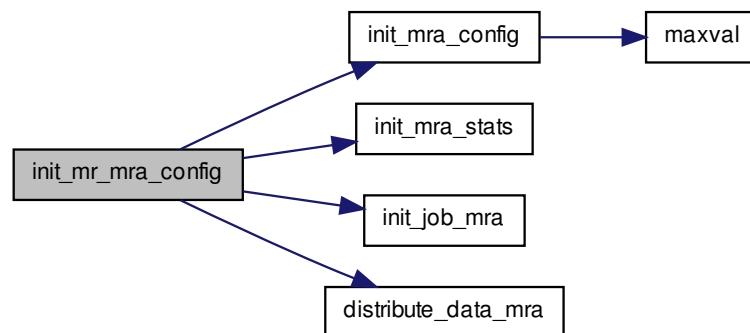
4.9.2.4 static void init_mr_mra_config (const char * *mra_config_file*) [static]

Initialize the MapReduce configuration.

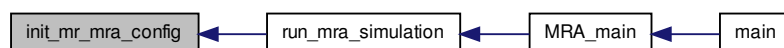
Parameters

<i>mra_config_file</i>	The path/name of the configuration file.
------------------------	--

Here is the call graph for this function:



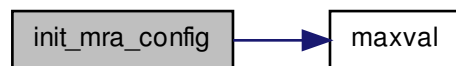
Here is the caller graph for this function:



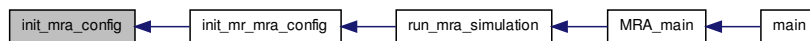
4.9.2.5 static void init_mra_config (void) [static]

Initialize the config structure.

Here is the call graph for this function:



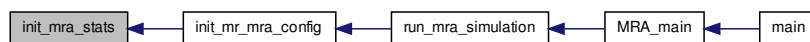
Here is the caller graph for this function:



4.9.2.6 static void init_mra_stats (void) [static]

Initialize the stats structure.

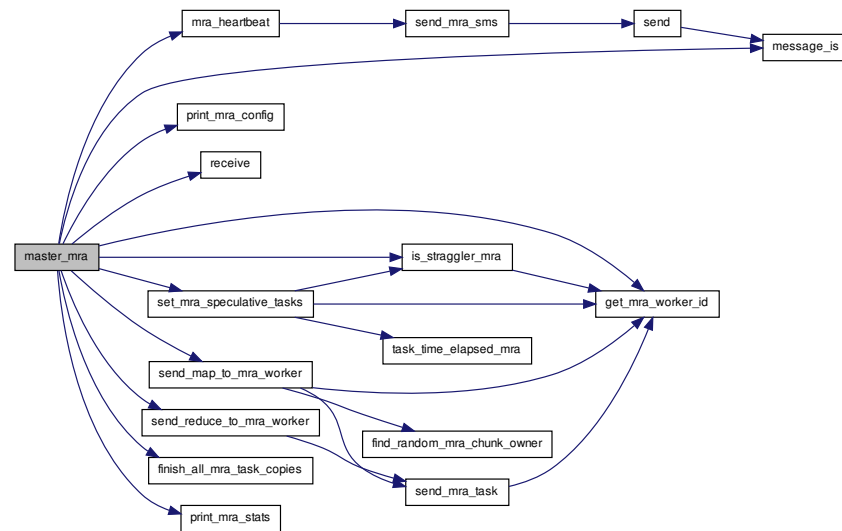
Here is the caller graph for this function:



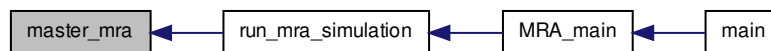
4.9.2.7 int master_mra (int argc, char * argv[])

Main master function.

Here is the call graph for this function:

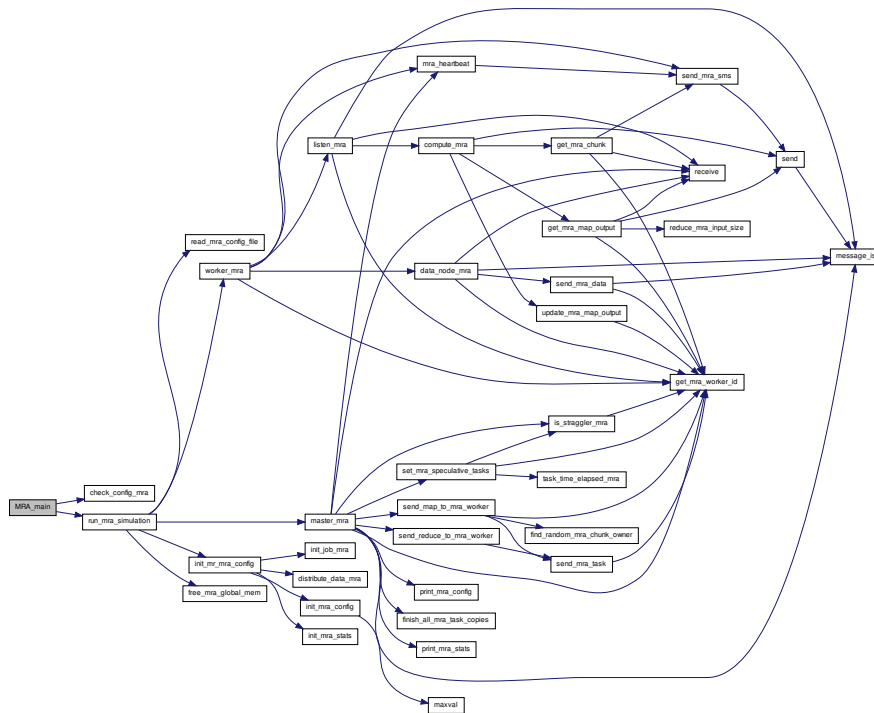


Here is the caller graph for this function:

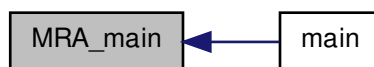


4.9.2.8 int MRA_main (const char * *plat*, const char * *depl*, const char * *conf*)

Here is the call graph for this function:



Here is the caller graph for this function:



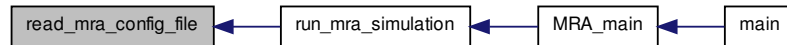
4.9.2.9 static void read_mra_config_file (const char * *file_name*) [static]

Read the MapReduce configuration file.

Parameters

<i>file_name</i>	The path/name of the configuration file.
------------------	--

Here is the caller graph for this function:

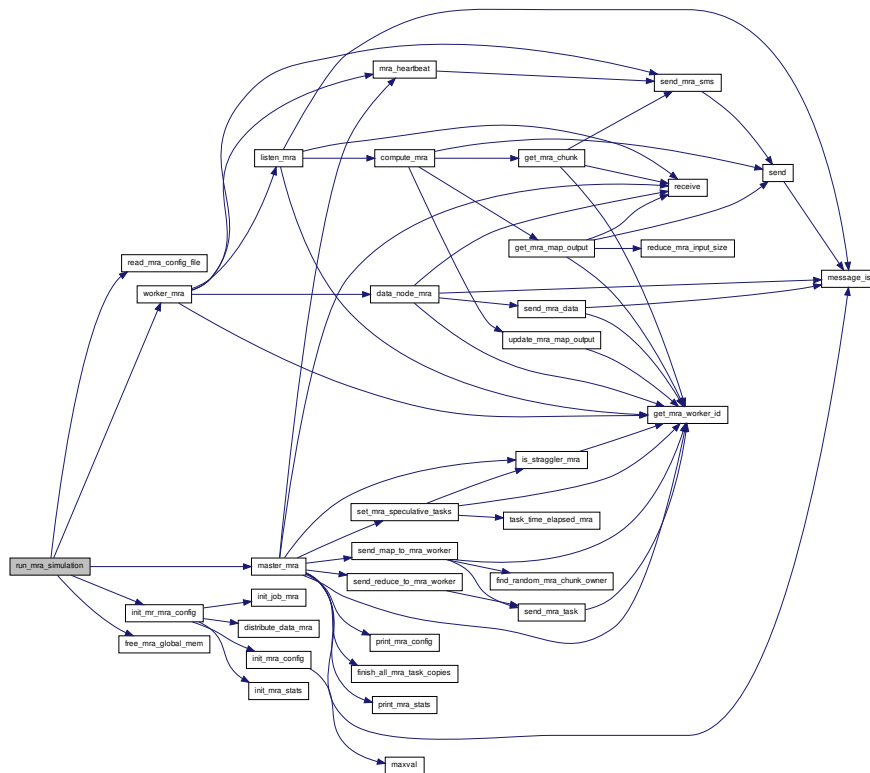


4.9.2.10 `static msg_error_t run_mra_simulation (const char * platform_file, const char * deploy_file, const char * mra_config_file)` [static]

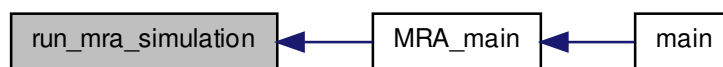
Parameters

<i>platform_file</i>	The path/name of the platform file.
<i>deploy_file</i>	The path/name of the deploy file.
<i>mra_config_file</i>	The path/name of the configuration file.

Here is the call graph for this function:



Here is the caller graph for this function:

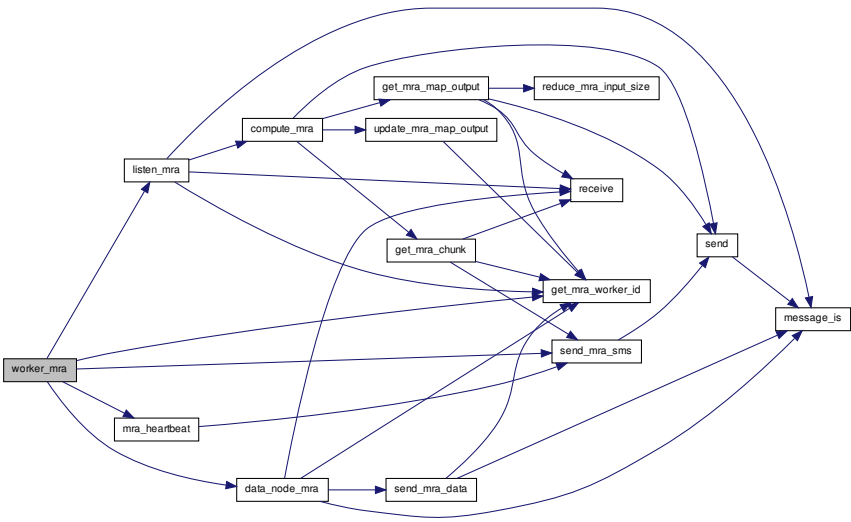


4.9.2.11 int worker_mra (int argc, char * argv[])

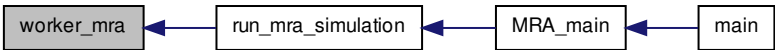
Main worker function.

This is the initial function of a worker node. It creates other processes and runs a mra_heartbeat loop.

Here is the call graph for this function:



Here is the caller graph for this function:

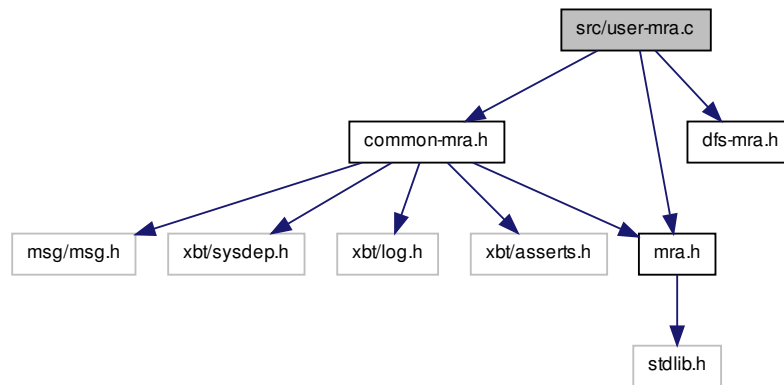


4.9.2.12 XBT_LOG_NEW_DEFAULT_CATEGORY (msg_test , "MRA")

4.10 src/user-mra.c File Reference

```
#include "common-mra.h"    #include "dfs-mra.h"    #include
```

"mra.h" Include dependency graph for user-mra.c:



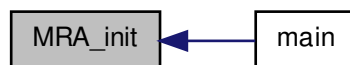
Functions

- void [MRA_init](#) (void)
- void [MRA_set_task_mra_cost_f](#) (double(*f)(enum [phase_e](#) phase, size_t tid, size_t wid))
- void [MRA_set_dfs_f](#) (void(*f)(char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas))
- void [MRA_set_map_mra_output_f](#) (int(*f)(size_t mid, size_t rid))

4.10.1 Function Documentation

4.10.1.1 void MRA_init (void)

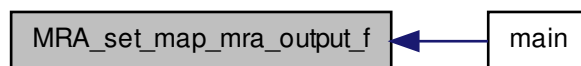
Here is the caller graph for this function:



4.10.1.2 void **MRA_set_dfs_f** (void(*) (char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas) f)

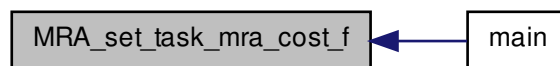
4.10.1.3 void **MRA_set_map_mra_output_f** (int(*) (size_t mid, size_t rid) f)

Here is the caller graph for this function:



4.10.1.4 void **MRA_set_task_mra_cost_f** (double(*) (enum phase_e phase, size_t tid, size_t wid) f)

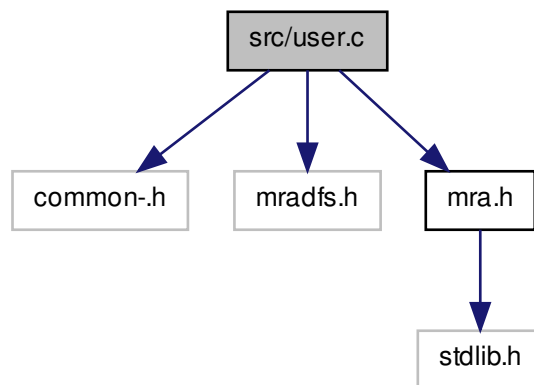
Here is the caller graph for this function:



4.11 src/user.c File Reference

```
#include "common-.h" #include "mradsfs.h" #include "mra.h" ×
```

Include dependency graph for user.c:



Functions

- void [MRA_init](#) (void)
- void [MRA_set_task_mra_cost_f](#) (double(*f)(enum [phase_e](#) phase, size_t tid, size_t wid))
- void [MRA_set_dfs_f](#) (void(*f)(char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas))
- void [MRA_set_map_mra_output_f](#) (int(*f)(size_t mid, size_t rid))

4.11.1 Function Documentation

4.11.1.1 void MRA_init (void)

Here is the call graph for this function:



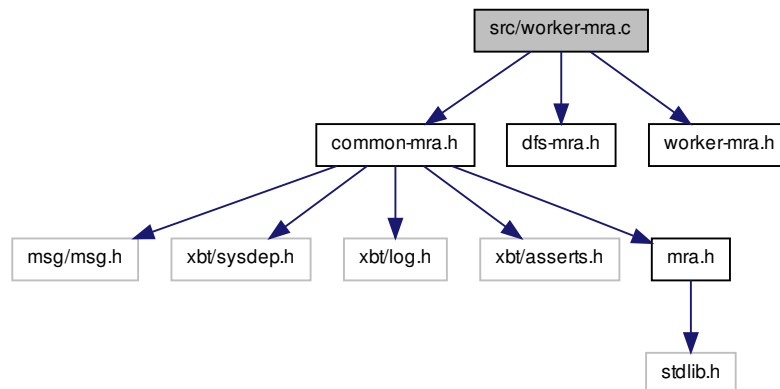
4.11.1.2 void MRA_set_dfs_f (void(*) (char **mra_dfs_matrix, size_t chunks, size_t workers_mra, int replicas) f)

4.11.1.3 void MRA_set_map_mra_output_f (int(*) (size_t mid, size_t rid) f)

4.11.1.4 void MRA_set_task_mra_cost_f (double(*) (enum phase_e phase, size_t tid, size_t wid) f)

4.12 src/worker-mra.c File Reference

```
#include "common-mra.h"    #include "dfs-mra.h"    #include
"worker-mra.h" Include dependency graph for worker-mra.c:
```



Functions

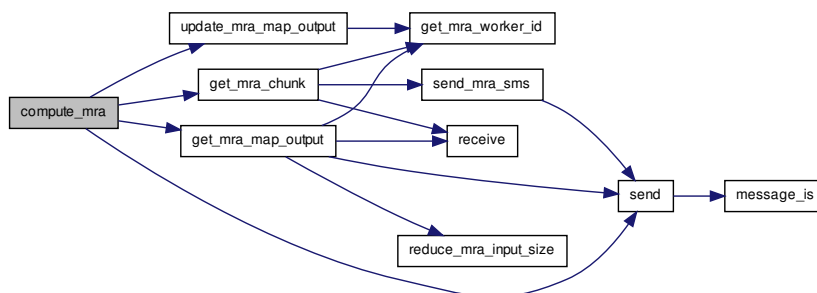
- [XBT_LOG_EXTERNAL_DEFAULT_CATEGORY](#) (msg_test)
- static void [mra_heartbeat](#) (void)
The mra_heartbeat loop.
- static int [listen_mra](#) (int argc, char *argv[])
Process that listens for tasks.
- static int [compute_mra](#) (int argc, char *argv[])
Process that computes a task.
- static void [update_mra_map_output](#) (msg_host_t worker, size_t mid)
Update the amount of data produced by a mapper.
- static void [get_mra_chunk](#) (mra_task_info_t ti)
Get the chunk associated to a map task.
- static void [get_mra_map_output](#) (mra_task_info_t ti)
Copy the intermediary pairs for a reduce task.
- size_t [get_mra_worker_id](#) (msg_host_t worker)
Get the ID of a worker.
- int [worker_mra](#) (int argc, char *argv[])
Main worker function.

4.12.1 Function Documentation

4.12.1.1 static int compute_mra (int argc, char * argv[]) [static]

Process that computes a task.

Here is the call graph for this function:



Here is the caller graph for this function:



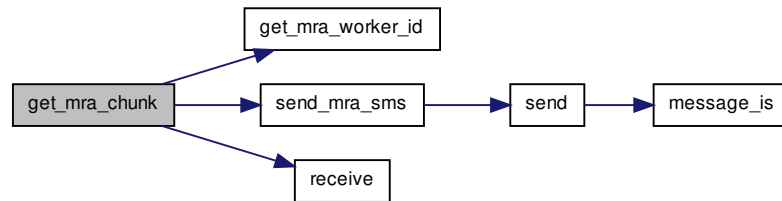
4.12.1.2 static void `get_mra_chunk (mra_task_info_t ti)` [static]

Get the chunk associated to a map task.

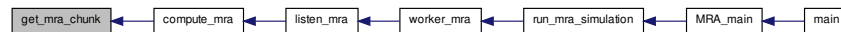
Parameters

<i>ti</i>	The task information.
-----------	-----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



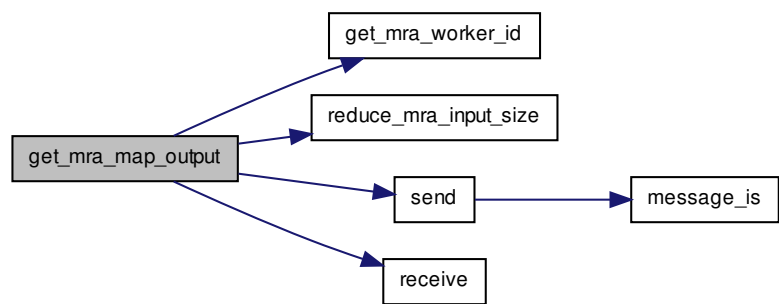
4.12.1.3 static void `get_mra_map_output (mra_task_info_t ti)` [static]

Copy the intermediary pairs for a reduce task.

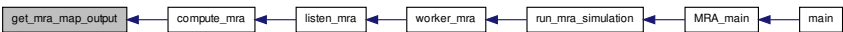
Parameters

<i>ti</i>	The task information.
-----------	-----------------------

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.1.4 `size_t get_mra_worker_id (msg_host_t worker)`

Get the ID of a worker.

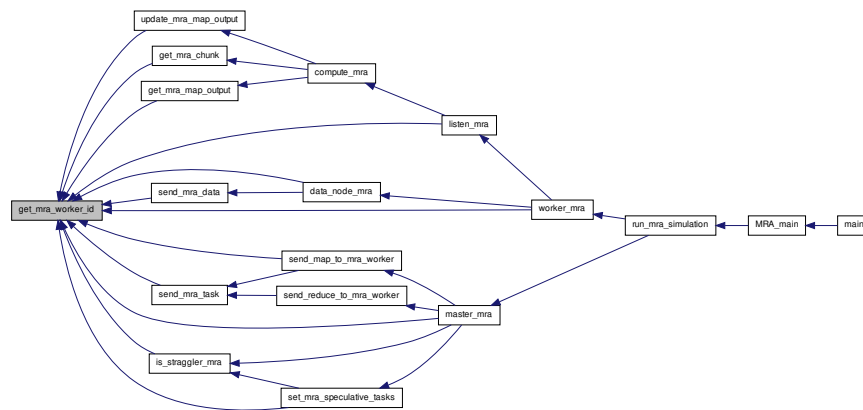
Parameters

<code>worker</code>	The worker node.
---------------------	------------------

Returns

The worker's ID number.

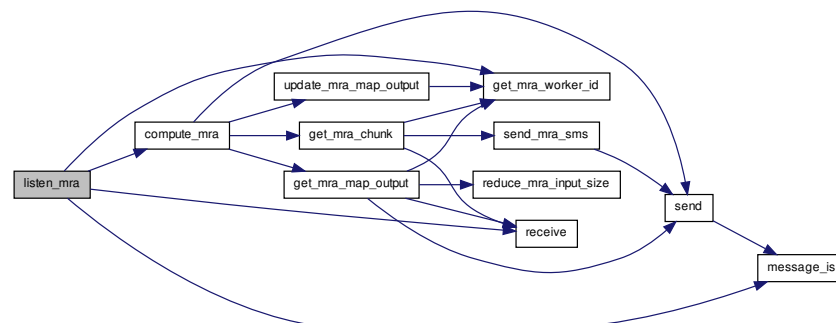
Here is the caller graph for this function:



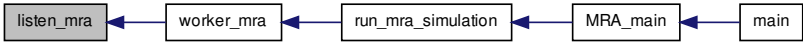
4.12.1.5 static int listen_mra (int argc, char * argv[]) [static]

Process that listens for tasks.

Here is the call graph for this function:



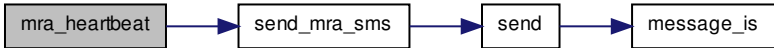
Here is the caller graph for this function:



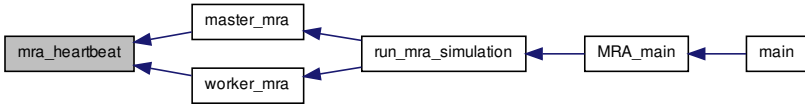
4.12.1.6 static void mra_heartbeat (void) [static]

The mra_heartbeat loop.

Here is the call graph for this function:



Here is the caller graph for this function:



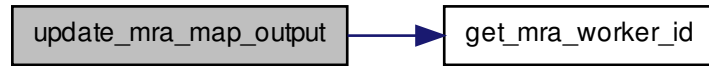
4.12.1.7 static void update_mra_map_output (msg_host.t worker, size.t mid) [static]

Update the amount of data produced by a mapper.

Parameters

<i>worker</i>	The worker that finished a map task.
<i>mid</i>	The ID of map task.

Here is the call graph for this function:



Here is the caller graph for this function:

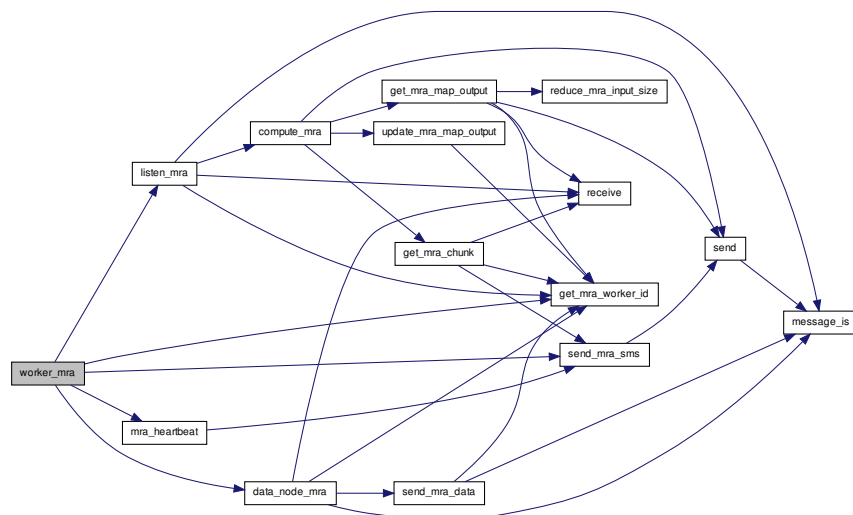


4.12.1.8 int worker_mra (int argc, char * argv[])

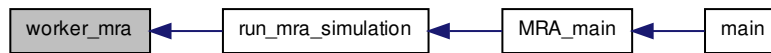
Main worker function.

This is the initial function of a worker node. It creates other processes and runs a mra_heartbeat loop.

Here is the call graph for this function:



Here is the caller graph for this function:



4.12.1.9 XBT_LOG_EXTERNAL_DEFAULT_CATEGORY (msg_test)