ADR: Dynamic-Aware RMS is Built on Application Information

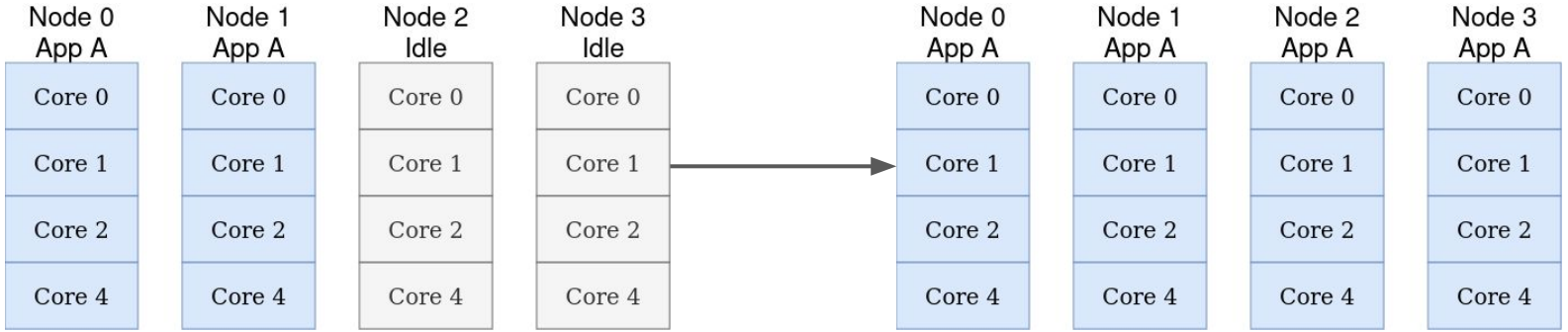
Authors

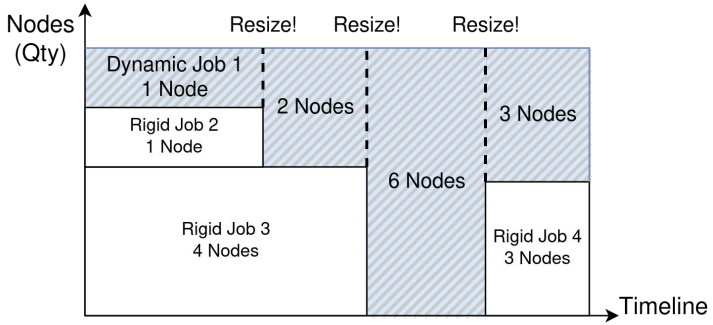
Iker Martín-Álvarez

dynres.readthedocs.io

Fix in ADR

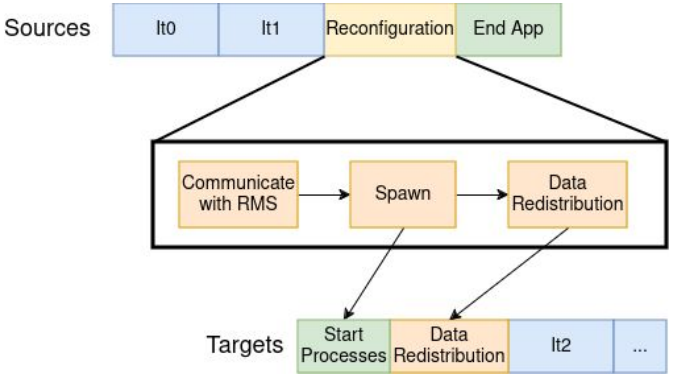
What is Dynamic Resource Management (DRM)?



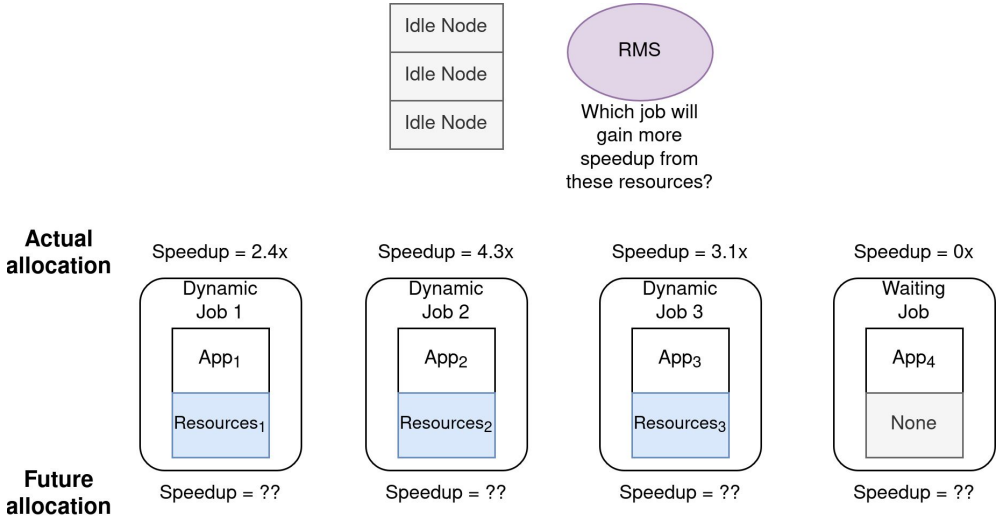


Challenges

- Scheduler must be capable of adapting allocation of executing jobs
- Scheduler requires collaboration from jobs to make informed decisions
- Applications require modifications to introduce reconfigurations

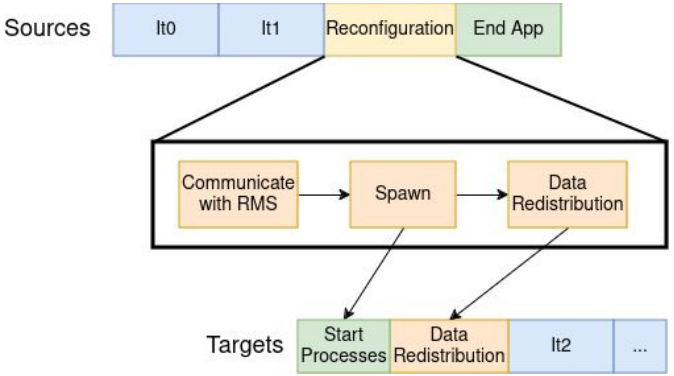


Example for reducing job running time

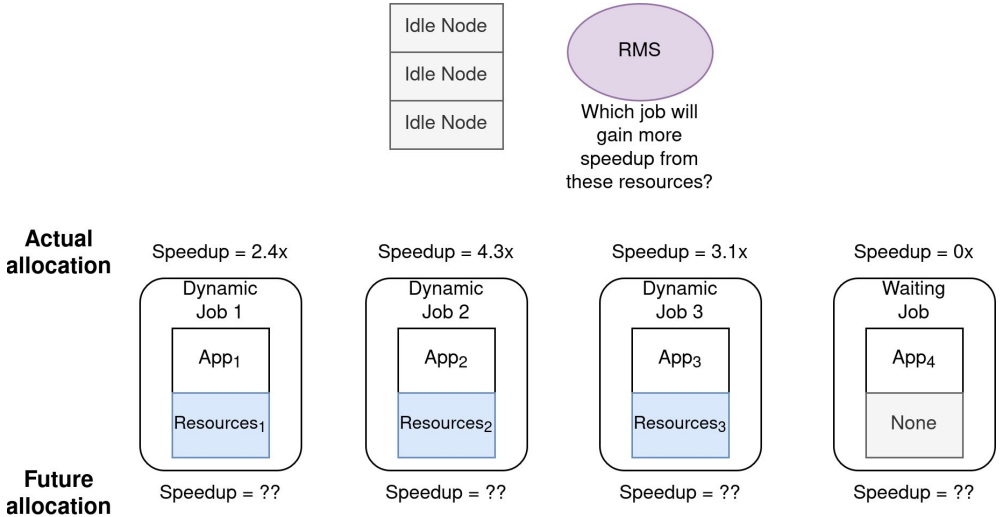


Challenges

- Scheduler must be capable of adapting allocation of executing jobs
- Scheduler requires collaboration from jobs to make informed decisions
- Applications require modifications to introduce reconfigurations

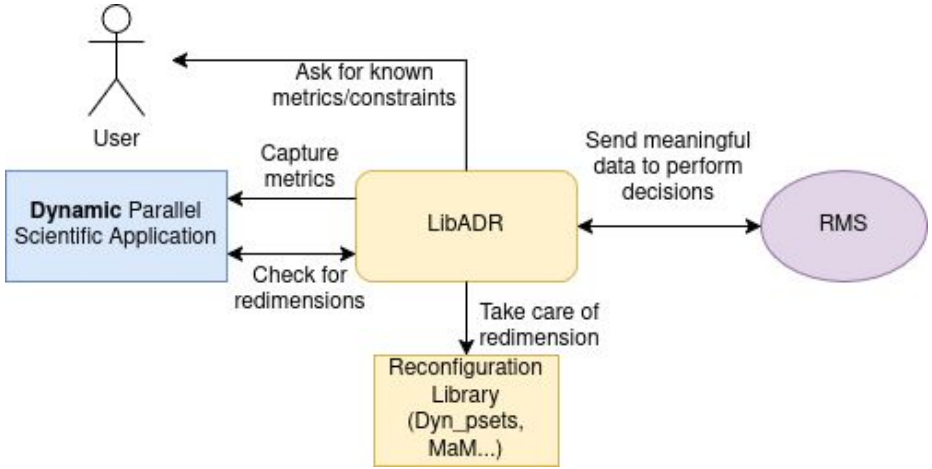


Idle resources, who gets them?



LibADR

- Awesome Dynamic Resources (ADR)
- Motivation: Reduce complexity for converting applications into dynamic
- Key: Able of capturing performance metrics and convert them into meaningful information for the RMS
- Build on top of DPP



Basic functions

```
ADR_init_pset();
ADR_adapt_check(state);
ADR_commit(state);
ADR_finalize_pset();
```

ADR repository

ADR - Basic Interface

```
1   int ADR_init_pset_from_mpi_session(
2       const MPI_Session i_mpi_session,
3       const char *i_custom_pset_name,
4       ADR_Pset *o_pset,
5       ADR_TYPE_BOOL *o_is_dynamic
6   );
7
8   int ADR_get_comm_from_pset(
9       const ADR_Pset i_pset,
10      MPI_Comm *o_comm
11  );
12
13  int ADR_finalize_pset(
14      ADR_Pset *io_pset
15  );
```

- ADR Pset is a handle, does not hold information.
- The boolean “is_dynamic” to detect the rank is a dynamic rank.
- An additional function is used to obtain the Communicator, which can change after resizes.

ADR - Reconfiguration Interface

```
1   int ADR_adapt_check(
2       const ADR_Pset i_pset,
3       ADR_TYPE_BOOL *o_migrate
4   );
5
6   int ADR_get_resize_hints(
7       const ADR_Pset i_pset,
8       ADR_Resize *o_resize_info
9   );
10
11  int ADR_rank_is_terminating(
12      const ADR_Pset i_pset,
13      ADR_TYPE_BOOL *o_terminate
14  );
15
16  int ADR_commit(
17      const ADR_Pset i_pset,
18      ADR_TYPE_BOOL *o_terminate
19  );
20
21  typedef struct ADR_Resize_struct {
22  int action; // Expand, Reduce, or combination
23  int local_state, *all_states;
24  MPI_Comm resize_comm;
25  } ADR_Resize;
```

- Variable “migrate” indicates if the user has to perform data redistribution.
- Hints are used during redistribution to know what changes will be performed at ranks level.
- After redistribution confirm with Commit to ensure it has ended.
- Variable “state” shows if the rank is obsolete, it continues or has emerged from the reconfiguration.

ADR - Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include 'adr.h'
int main() {
    int iteration_total = 8;
    int iteration_current_id = 0;

    int is_dynamic;
    int terminate_rank = ADR_FALSE;
    MPI_Session mpi_session;
    MPI_Comm comm;
    MPI_Session_init(MPI_INFO_NULL, MPI_ERRORS_ARE_FATAL,
     &mpi_session);

    ADR_Pset adr_pset;
    ADR_init_pset_from_mpi_session(mpi_session, 'mpi://WORLD',
     &adr_pset, &is_dynamic);
    if(ADR_PSET_NULL == adr_pset) {
        MPI_Session_finalize(&mpi_session);
        return 0;
    }

    if (!is_dynamic) {
        // Init APP
    } else {
        // User Data Redistribution
        ADR_commit(adr_pset, &terminate_rank);
    }
    ADR_get_comm_from_pset(adr_pset, &comm);

    for (; iteration_current_id < iteration_total &&
     !terminate_rank; iteration_current_id++) {
        // COMPUTATION
        terminate_rank = check_reconf(adr_pset, &comm,
         &problem_size, &iteration_current_id);
    }

    ADR_finalize_pset(&adr_pset);
    if(MPI_COMM_NULL != comm) { MPI_Comm_disconnect(&comm); }
    MPI_Session_finalize(&mpi_session);
    return 0;
}
```

```c
int check_reconf(ADR_Pset adr_pset, MPI_Comm *comm, int
  *problem_size, int *current_iter) {
    int migrate_data = ADR_FALSE;
    int terminate_rank = ADR_FALSE;

    ADR_adapt_check(adr_pset, &migrate_data);
    if (migrate_data)
    {
        // User Data Redistribution
        ADR_commit(adr_pset, &terminate_rank); // Indicates to ADR
          that data migration has been carried out
        if(MPI_COMM_NULL != *comm) { MPI_Comm_disconnect(comm); }
        if (!terminate_rank) ADR_get_comm_from_pset(adr_pset,
          comm);
        user_set_opti_info(adr_pset);
    }
    return terminate_rank;
}
```

ADR - COL Objects

```
1   enum adr_model_kinds{ADR_MODEL_CONSTRAINTS,
      ADR_MODEL_PREFERENCES, ADR_MODEL_GENERAL, ADR_MODEL_AMDAHL,
      ADR_MODELS_SIZE};
2   enum adr_model_constraints{ADR_CONSTRAINT_PROC_LOWER_LIMIT,
      ADR_CONSTRAINT_PROC_UPPER_LIMIT, ADR_CONSTRAINT_PROC_POWER2,
      ADR_CONSTRAINT_PROC_DIVISIBLE2, ADR_CONSTRAINT_MAPPING,
      ADR_CONSTRAINT_MEM_LOWER_LIMIT, ADR_CONSTRAINTS_SIZE};
3   enum adr_model_preferences{ADR_PREFERENCE_PROC_SWEETSPOT,
      ADR_PREFERENCE_PROC_POWER2, ADR_PREFERENCE_PROC_DIVISIBLE2,
      ADR_PREFERENCE_MAPPING, ADR_PREFERENCE_CHECK_ITERS,
      ADR_PREFERENCES_SIZE};
4   enum adr_model_general{ADR_MODEL_STRUCT, ADR_GENERAL_SIZE};
5   enum adr_model_amdahl{ADR_AMDAHL_STRUCT, ADR_AMDAHL_SIZE};
6
7
8   typedef struct {
9   int qty;
10  int *procs;
11  double *speedup;
12  } ADR_Model_general;
13
14  typedef struct {
15  double sequential_part, parallel_part;
16  } ADR_Model_amdahl;
```

- COL objects help the RMS in taking a decision for the job. Three different types:
  - Constraints: Requirements of the application.
  - Preferences: Allocations that are preferable by the application.
  - Model: Characterizes the application metrics.
- Set function allows to set each of the types or change them. The compute creates the COL object to send to the RMS.
- The model characterizes the metrics of the application.

ADR - COL Objects

```
1       int ADR_pset_set_opti_information(
2           const ADR_Pset i_pset,
3           ADR_Info i_info,
4           ADR_Info_kind i_info_kind
5       );
6
7       int ADR_pset_compute_opti_information(
8           const ADR_Pset i_pset
9       );
10
11      int ADR_pset_opti_information_requested(
12          const ADR_Pset i_pset,
13          ADR_TYPE_BOOL *requested
14      );
```

- COL objects help the RMS in taking a decision for the job. Three different types:
  - Constraints: Requirements of the application.
  - Preferences: Allocations that are preferable by the application.
  - Model: Characterizes the application metrics.
- Set function allows to set each of the types or change them. The compute creates the COL object to send to the RMS.
- The model characterizes the metrics of the application.

ADR - COL Objects Example

```c
void user_set_opti_info(ADR_Pset adr_pset) {
    ADR_Info info_c, info_p, info_m;
    ADR_Model_general basic = ADR_MODEL_GENERAL_INIT; // Copy all values to default

    ADR_info_create(&info_c, ADR_MODEL_CONSTRAINTS);
    int lower_limit, upper_limit, div2;
    lower_limit = 2;
    upper_limit = 6;
    div2 = 1;
    ADR_info_set(info_c, ADR_CONSTRAINT_PROC_LOWER_LIMIT, &lower_limit);
    ADR_info_set(info_c, ADR_CONSTRAINT_PROC_UPPER_LIMIT, &upper_limit);
    ADR_info_set(info_c, ADR_CONSTRAINT_PROC_DIVISIBLE2, &div2);

    ADR_info_create(&info_p, ADR_MODEL_PREFERENCES);
    int iter_limit = 1; // One iter have to pass before checking with the RMS
    ADR_info_set(info_p, ADR_PREFERENCE_CHECK_ITERS, &iter_limit);

    ADR_info_create(&info_m, ADR_MODEL_GENERAL);
    int qty = 4;
    int procs[4] =      {1,   2,    8,   24};
    double speedup[4] = {1, 1.43, 6.4, 15.1};
    basic.qty = qty;
    basic.procs = procs;
    basic.speedup = speedup;
    ADR_info_set(info_m, ADR_MODEL_STRUCT, &basic);

    ADR_pset_set_opti_information(adr_pset, info_c, ADR_MODEL_CONSTRAINTS);
    ADR_pset_set_opti_information(adr_pset, info_p, ADR_MODEL_PREFERENCES);
    ADR_pset_set_opti_information(adr_pset, info_m, ADR_MODEL_GENERAL);
    ADR_pset_compute_opti_information(adr_pset);
    ADR_info_free(&info_c);
    ADR_info_free(&info_m);
}
```

Resulting code in the RMS to test different resource allocations in the application.

```python
def adr_speedup(proc_amount):
    if proc_amount < 2:
        return 0
    procs=[1,2,8,24,]
    speed=[1.000000,1.430000,6.400000,15.100000,]
    if proc_amount in procs:
        return speed[procs.index(proc_amount)]
    px=[0.732394,0.180336,0.090307,-0.003037,]
    sum=0
    for i in range(len(px)):
        sum+=px[i]*proc_amount**i
    return sum
```

Hands-On - Basic Code

Start the cluster:
./docker-cluster.sh --docker_repo=hawkmooneternal
--env_install=tutorial_dynreshpc26 --num_nodes=9 --drop_in_host=n1

Start the environment:
dynpkgs env_deploy tutorial_dynreshpc26

Enter the libADR project:
cd /opt/hpc/build/libadr/examples

There is a simple iterative application to see how the information for deciding reconfigurations is transmitted to the RMS.
Can be executed with:
python3 run_test_dynrm.py --topology_file=2_node_system.yaml
--submission_file=example_basic.batch --verbosity=9 --output_dir=expansion_test
--step_size=single

ADR: Dynamic-Aware RMS is Built on Application Information

Authors

Iker Martín-Álvarez (martini@uji.es)