Data-aware Scheduling User Guide

Platform Symphony
Version 5.1
April 2011

Platform Computing

Contents

Data-aware scheduling

Data-aware scheduling is a feature of the Data Affinity add-on product. Data Affinity reduces network traffic between hosts in the cluster and allows for more flexible scaling of data-intensive applications with improved performance.The data-aware scheduling feature allows Platform Symphony to intelligently schedule application tasks and improve performance by taking into account data location when dispatching the tasks. By directing tasks to resources that already contain the required data, application runtimes can be significantly reduced. As well, this feature can help to meet the challenges of latency requirements for real time applications.

An additional software license is required to use this feature in a grid environment. Use of this feature with Symphony DE does not require a license. This feature is packaged with Symphony and does not require separate deployment.

Scope

Applicability	Details
Operating system	• Windows • Linux • Solaris

Feature licensing

Follow these steps when you want to add or update a license for the data-aware scheduling feature.

1. Acquire a data-aware scheduling license key from Platform.
2. Append the license key to $EGO_CONFDIR/license.dat. Note that all management hosts running an SSM must have access to this license file.
3. Set schedulingAffinity="DataAware" in the application profile. Configure the plug-in command, if applicable.
4. Register and enable the new application profile.

Note:

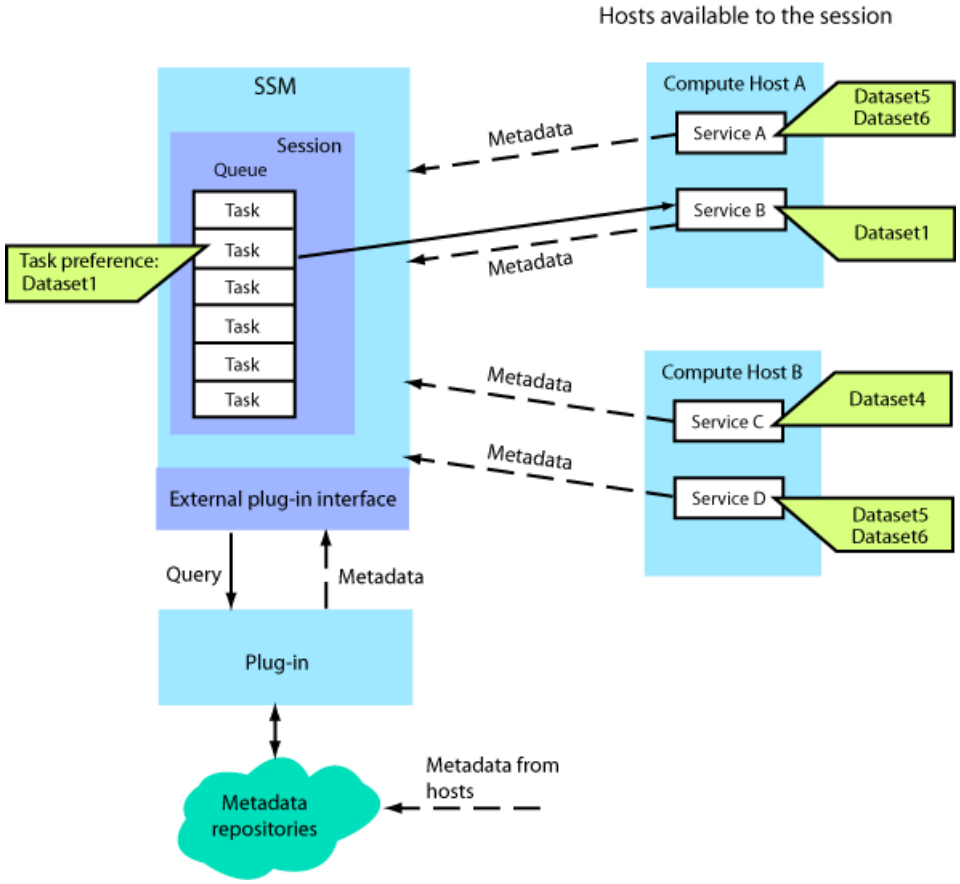
If your application was already configured and registered before you acquired the license, you need to disable and re-enable the application so that the license is verified and the feature is enabled.

About data-aware scheduling

Workload schedulers focus on dispatching tasks to compute hosts and transferring data either directly to the compute hosts or delegating data retrieval to the service. The time required for this data transfer from various sources to where the work is being processed can lead to inefficient use of CPU cycles and underutilization of the resource. With the data-aware scheduling feature, you can specify a preferential association between a task and a service instance or host that already possesses the data required to process the workload. This association is based on the evaluation of a user-defined expression containing data attributes capable of being collected. The evaluation of this expression is carried out against the data

attributes of each service instance available to the session. Typically, a data attribute is an identifier for a dataset that is already available to a service instance before it processes workload.

The following example illustrates the concept of data-aware scheduling at the task level. The data preference expression has been evaluated and it is determined that a task in the queue prefers to run on a service instance where the service instance already possesses *Dataset1*. The SSM collects metadata (service attributes) from all the resources available to the session at that moment. *Service B* with *Dataset1* is currently available and since it is the best match for that task according to the specified preference, the task is subsequently dispatched to Service B.



Expressing a preference for data

Specifying data attributes in a preference expression

Attributes take the form of identifiers and can be specified with the +, -, *, and / operators within an expression. An example of an expression would be "DataSet1 + DataSet2", meaning that it is preferred to send workload to a service instance that possesses DataSet1 and DataSet2. The operators can also be used to normalize disparate terms such as data and memory or to give weight to specific terms.

Attribute names have a 32 character limit and can only contain alphanumeric and underscore characters; if you want to use data attributes with names that do not comply with these rules, aliases must be defined. A resource attribute definition can be used to define an alias and to override the default value for an attribute should the session level default or system default be inappropriate.

How Symphony handles the result of the expression

The result of each expression is a numeric value that is obtained by applying the operators to the attributes in the expression. If the preferred data is available to a service, it should programmatically publish a value for the attribute. Alternatively, the value of the attribute may be collected from the data-aware scheduling plug-in, if present. Once the result is obtained for each resource being evaluated, it is used to sort the resources in ascending order. This means the resource that evaluates to the lowest value is the most preferred.

When no information is available for a resource attribute that is involved in a resource evaluation, the resolution of the expression still proceeds. In such cases, Symphony attempts to substitute a default value for each attribute that it cannot resolve. The value of the attribute is resolved by the system in the following manner, in the given order:

1. attempt to find any published or collected value
2. retrieve the current default for the alias (if defined)
3. retrieve the current default for the session (if defined)
4. retrieve the system default (1.0E+300)

How preference affects dispatch order

Once a preference is specified on a task within a session, the best match for that task with every service instance currently assigned to the session is considered during the dispatching of the task. This means that once a service becomes available, it is given the next task with the best match for that service instance at the current moment. It is important to note that the next task to be dispatched may not currently be at the front of the queue within the session, i.e., the order of task dispatch depends on the currently available service instances and the preference associated with the tasks and not on the order that tasks are submitted to the session.

Note that the data-aware scheduling feature does not affect the behavior of tasks with the PriorityTask setting enabled. Those tasks are still dispatched before other tasks.

Factoring the cost of data transfer

When you want to specify a preference for specific data, the expectation is that attributes in the expression would have their default value set to some number representing the cost of moving this data to that resource.

For example, if we consider the case where a task within a session requires datasets Dataset1, Dataset2 and Dataset3, the preference for the task could be represented as "Dataset1 + Dataset2 + Dataset3". Since expressions are evaluated and sorted in ascending order, we would expect the most preferred resource for this task would have the lowest cost to get the data that the task requires to execute.

Since a service instance is typically able to inform the middleware about what resource attributes (i.e. data sets) it has access to, a fixed value can be substituted for any missing attributes when evaluating the task preference against this service instance. If you use the same fixed value to represent each missing attribute, the system will assign each missing data set the same cost so service instances that have access to the most data sets would be the most preferred and so on. But in reality, it is not generally true that all missing data sets would have the same cost to acquire, especially in the case where the data sets can vary greatly in size or be located on hosts with disparate network topologies and network access speeds. That is why to gain the most benefit from the data-aware scheduling feature, the default value for a data attribute should be carefully considered when applied to a preference expression to ensure that it is representative

in some way of the cost of retrieving the missing data for the service instance. This default value is set through the resource attribute definition API.

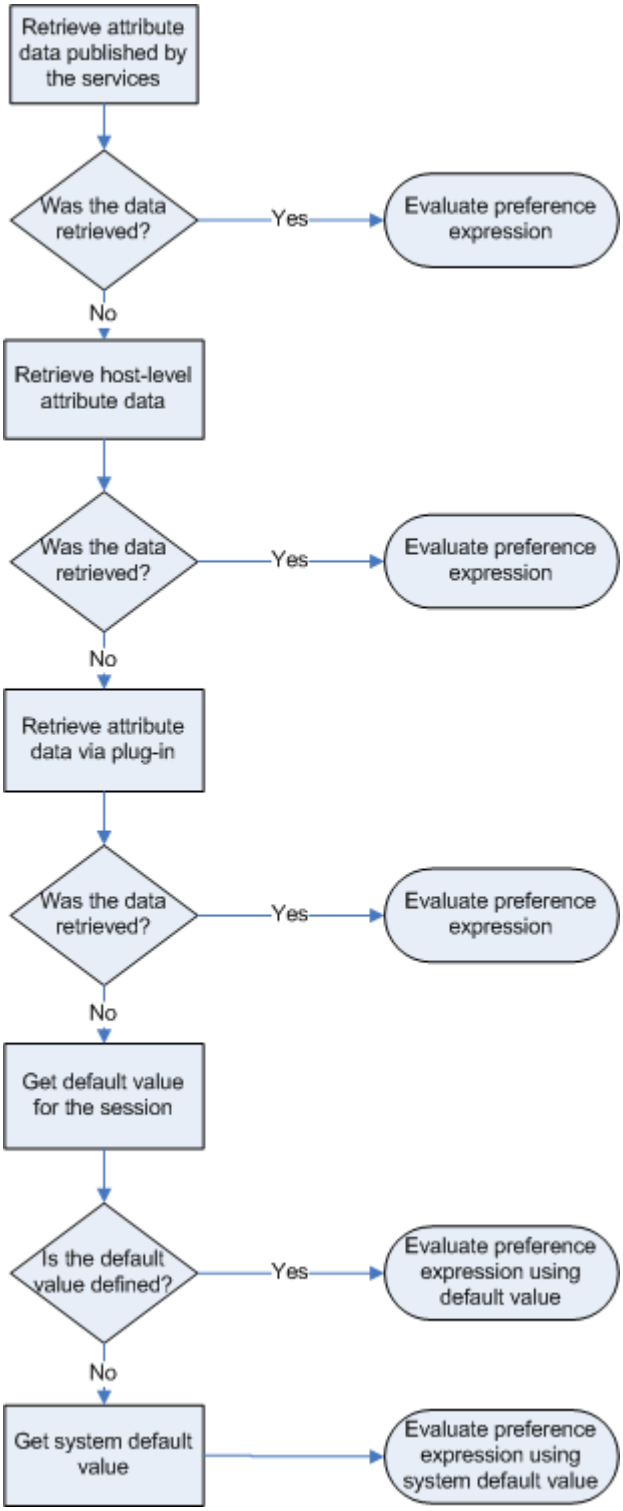
Data-aware scheduling plug-in

This direct plug-in provides an additional way for Symphony to obtain the location and cost data associated with preferred resources. The plug-in enhances data-aware scheduling by allowing Symphony to get this data not only from the service side publishing/subscription mechanism, but also from a direct call to a centralized process that can be customized for integration with 3rd party products.

The plug-in process requests the information from an external application and returns it back to the SSM to be used by Symphony as the preferred scheduling location and value for the attribute in the same way as if it were published by a service. Sample code for the plug-in process is provided in an appendix at the end of this document.

To improve performance, this feature also provides caching for the result received from the plug-in so subsequent calls for the same attribute will return cached values.

Here is the sequence the SSM follows to obtain the resource attribute value when data-aware scheduling is enabled.



If the plug-in is configured and the preferred attributes do not have any published values during the current evaluation cycle, the SSM requests locations and values from the plug-in. The following describes the functional flow when the plug-in is used to retrieve resource attribute data for a task.

1. A task is submited with one or more preference attributes.
2. Upon the task's submission, the SSM parses the preferences and puts the task in the pending queue.
3. The preferences are evaluated and the SSM tries to find values for the corresponding attributes. Since, in this case, the attribute was not published by the service, the expression is not resolved.
4. The SSM checks if the direct plug-in is configured for the application. The direct plug-in process is started using a startup command defined in the application profile.
5. The SSM repeatedly calls the plug-in interface with the attribute names to retrieve the corresponding locations and values.
6. Tasks are dispatched to currently available resources for the session that are the best match according to the preferred attributes.

Configuring data-aware scheduling

Enabling data-aware scheduling for tasks

Data-aware scheduling is enabled at the application level with the schedulingAffinity attribute in the Consumer element of the application profile. When the attribute is set to DataAware, the SSM collects data attributes of service instances and hosts and evaluates them against a user-defined preference expression. Note that setting the attribute to DataAware automatically enables resource-aware scheduling. When the attribute is set to None (default), no metadata is collected by the SSM and no preference is applied.

Example:

```
<Consumer applicationName="SharingDataCPP" ...schedulingAffinity="DataAware" />
```

The schedulingAffinity attribute can be configured through the PMC or by manually editing the application profile.

Configuring the plug-in

The data-aware scheduling plug-in is configured using the following attributes in the application profile.

- SSMResPubPluginCmd

  Specifies a command line for starting the plug-in process , one per Symphony application, in the application profile.
- SSMResPubCacheEnabled

  Sets data caching on or off. The default is on.

Here is an example of the attributes configuration in the application profile.

```
<SSM resReq="" workDir="${EGO_SHARED_TOP}/soam/work"
 startUpTimeout="60" shutDownTimeout="300" SSMResPubPluginCmd="C:\extplugin.exe"
  SSMResPubCacheEnabled="false">
</SSM>
```

Client API

Configuring data-aware scheduling for tasks is only possible through the client-side API. For more information about the API, refer to the API reference documentation.

Defining default attribute values for the session

You can define a substitute value for attributes in a preference expression in the event that those attributes are unknown at the time of evaluation. If this default value is not specified, the system default, i.e., 1.0E +300, is substituted.

Example:

SessionCreationAttributes.setDefaultResourceAttributeValue (float)

Specifying data preference

You can programmatically describe data preference for resources.

- ResourcePreference()

  This method creates an empty preference that can be populated by data attribute definitions.
- ResourcePreference(...)

  This method allows you to construct an expression for the data preference.

  Example:

  ResourcePreference ("Dataset1 + Dataset2")

  This expression means that it is preferred to run this workload on a service that has Dataset1 and Dataset2.

Creating attribute definitions

You are not required to use an attribute definition with your preference unless your application has the following requirements:

- The attribute in the expression has a different name than the attribute that will be published to the application.
- The attribute in the expression requires a default value that cannot be satisfied by either session-level or system default values.

Example:

```
ResourceAttributeDefinition A ("/dir/filenameA.dat", "A", 100);
ResourceAttributeDefinition B ("/dir/filenameB.dat", "B", 150);
```

Adding attribute definitions to a data preference

The AddDefinition(ResourceAttributeDefinition) method allows a definition to be associated with a data preference.

Example:

```
ResourceAttributeDefinition A ("/dir/filenameA.dat", "A", 100);
ResourceAttributeDefinition B ("/dir/filenameB.dat", "B", 150);
ResourcePreference rp ();
rp.addDefinition(A);
rp.addDefinition(B);
```

This code would result in the following data preference expression: "A + B".

As a best practice, you should call `addDefinition()` on the `SessionCreationAttributes` object at session creation time as it is more efficient than calling `addDefinition` on the `ResourcePreference` object that is associated with each task.

Associating a data preference with a task

A data preference is associated with a task by binding it to a TaskSubmissionAttributes object.

Example:

TaskSubmissionAttributes.setResourcePreference(ResourcePreference)

Service API

Publishing details about your datasets

The ResourceAttributes object allows you to publish details about resource attributes such as datasets that are available to the service instance. The application makes these details available to the middleware to help with optimizing the scheduling in the event the workload defines a preference that includes this attribute. Resource attributes can exist either at a service instance level or be generally available to any service instance on a host.

Example:

ResourceAttribute.setName(name)

ResourceAttribute.setValue(value)

ResourceAttribute.setScope(Host)

ServiceContext.publish(ResourceAttribute)

A service can unpublish resource attributes using the ServiceContext.unPublish(ResourceAttribute) method. When calling unPublish(), only the name of the resource attribute is required.

As a best practice, when you publish details about datasets that have been acquired by the service, you can treat the data as having no cost to acquire. This means a value of ‘0’ would be ideal for these types of attributes.

Configure data-aware scheduling

Data-aware scheduling is configured in the application profile. The following steps describe configuration using the Platform Management Console (PMC). To configure data-aware scheduling manually, set the appropriate attributes in the application profile as described in the data-aware scheduling feature reference.

1. In the PMC, click Symphony Workload > Configure Applications.

   The Applications page displays.
2. Select the application you want to modify.

   The Application Profile page displays.
3. Expand the Resource and Data Aware Scheduling section. In the Scheduling Affinity dropdown list, select Data Aware.

Note:

When Data Aware is configured, data-aware scheduling at the task level is enabled. This configuration also enables resource-aware scheduling at the session level. For more information about resource-

aware scheduling, refer to the Cluster and Application Management Guide.

4. If you are using a plug-in to collect data attributes, select true to enable the plug-in cache. In the textbox, enter the plug-in start command including the path.
5. In the Default Resource Attribute Value textbox, enter a value that will be substituted by the middleware if the preferred data attribute value is not available.
6. Click Save to apply your changes.

   Symphony reregisters the application for the changes to take effect.

Appendix: Data-aware Scheduling Plug-in Development

Protocol for interactions between the SSM and the plug-in process

This section describes the protocol that must be implemented by the plug-in code to interact with the SSM.

1. The protocol is ASCII and line-based with the <CR (\n)> character used as the line separator for portability between platforms and programming languages. Note that on Windows, CR='\r"\n' and on Unix (including Linux), CR='\n'.
2. The protocol implements synchronous communication with the SSM where the SSM initiates the request and the plug-in replies with a response. As a result, the response from any plug-in operation can be success or failure. To distinguish between success or failure, the very first number in any response line is either an error code (non-zero value) or 0 (successful response). An error code is followed by the accompanying error message, such as:

   "1 ERROR: bad initialization parameters.<CR>"
3. During the initial sequence, the SSM spawns the plug-in process and waits for the first initialization response. The plug-in process, starts up and after finishing its initialization (parameter for the initialization can be passed as command line arguments), replies to standard output on success:

   "Status version <error>" , status = 0 for success or non-zero for failure.

   "0 1.0.0.0"

   where the first number is success code 0 followed by the current protocol version supported by this plug-in (current version 1.0.0.0)

   -Have timeout to detect hang, default=30 seconds.
4. The SSM specifies request commands identified by one character (so protocol can be potentially extended up to 256 commands) followed by command arguments separated by a space character (' '). Currently the SSM provides only one command specified by character 'L' ("Location") followed by a space-separated list of attribute names.

   Example:

   "Command attr1 …attrN CR"

   where Command=L(return locations and Cost)

   "L attr1 attr2 attr3 attr4<CR>"
5. The plug-in process responds to the "Location" command with multiple response lines, one per attribute and in the same attributes order. Each successful response line starts with the success code 0 followed by (separated by space) the cost value and then one or more locations until the <CR>-character.

   Example:

   "Status Cost1 Location1 Cost 2 Location2…CostN LocationN CR"

   The following example assumes the values for four attributes are available:

"0 1.0 host1" for attr1

"0 2.0 host2 0.0 host3" for attr2

"0 3.123 host4 7 host5 10000 host6 0 host7" for attr3

"0 1.0 *" for attr4

The following example assumes the value for attr3 is not available to the plugin:

"0 1.0 host1" for attr1

"0 2.0 host2 0.0 host3" for attr2

"1 Do not have the value" for attr3; this value will be cached and returned in subsequent calls.

Note:

: The plug-in has to return number of response lines equal to number of attributes in the previous "Location" command and in the same order. The response can be a mix of successes and failures.

The asterisk (*) character for location means "any available host".

6. The plug-in should be ready at any time to get EOF character or error from standard input (e,g, "broken pipe"), which means a shutdown request from the SSM. In this case, the plug-in process must do appropriate cleanup and exit by itself, as the SSM will never kill the plug-in process.

Code sample

This section provides sample code for the data-aware scheduling plug-in process. Use the sample code as a template and add the necessary logic to retrieve the host location and transfer cost data from the metadata repositories.

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#ifndef WIN32
#include <unistd.h>
#endif
#include <errno.h>
static const char DELIMITER = ' ';
#ifndef WIN32
#define STDIN 0
#endif
int main(int argc, char* argv[])
{
    // reply initialization OK and plugin's protocol version number (1.0.0.0)
    fprintf(stdout, "0 1.0.0.0\n");
    fflush(stdout);
    // loop forever
#ifndef WIN32
    while(true)
    {
        fd_set fds;
        struct timeval tv;
        FD_ZERO(&fds);
        FD_SET(STDIN, &fds);
        tv.tv_sec = 0;
        tv.tv_usec = 100;
        int retVal = select (STDIN + 1, &fds, NULL, NULL, &tv);
        //select time out
        if (retVal == 0)
        {
            if (getppid() != 1)
            {
                continue;
            }
            //parent process is dead.
            else
            {
                break;
            }
        }
        //select error
        else if (retVal == -1)
        {
            break;
        }

        char line[1024]="";
        if (fgets(line, sizeof(line), stdin) == 0)
        {
            break;
        }
```

```
#else
    for(char line[1024]="";fgets(line, sizeof(line), stdin) != 0;line[0]=0 )
    {
#endif
 // trim <CR> in the end
        char* ptr = line+strlen(line)-1;
        if(*ptr == '\n')
            *ptr = '\0';
        // tokenize
        char* token = strtok( line, &DELIMITER);
        if( token == 0 )
        {
            fprintf(stdout,
                "3 ERROR: unknown command received by the plug-in process '%s'\n",
                argv[0]);
            fflush(stdout);
            continue;
        }
        switch( token[0] )
        {
        case 'L': // GET_LOCATION
            { // block

            int cnt=0;
            while( (token = strtok( 0, &DELIMITER)) != 0 )
            {
                ++cnt;
                 // return status=0, cost/location
                fprintf(stdout, "0 0 hosta 0 oparmar.noam.corp.platform.com\n");
                fflush(stdout);

            }
            if(!cnt)
            {
                // print error
                fprintf(stdout,
                "6 ERROR: cannot get any parameters for the command '%s'\n",
                line);
                fflush(stdout);
            }
            } // end of block
            break;
        case 'T': // GET_TOPOLOGY
           {
           }
```

```
            break;
        case 'Q': // QUIT

            return(0); // break pipe with parent
            break;

        default:  // unknown command!
            fprintf(stdout, "3 ERROR: unknown command received by the plug-in process
            '%s'\n", argv[0]);

            fflush(stdout);
        }
    }

    return(0); // break pipe with parent
}
```