The Sun Distributed Computing System (SDCS)

Jos van der Meer (VMX)
Johan van Veen (VMX)
Gertjan van Gent (Sun)
Maarten Westenberg (Sun)
Frans Bouma (VMX) [1]

VMX Professional Services
Sun Microsystems Nederland

ABSTRACT

In a network with (Sun) workstations, a lot of potential computing power is available. However, not every workstation is fully used at all time, in which case its computing power is wasted. The Sun Distributed Computing System (SDCS) is developed to make maximum use of all available resources in a network of workstations.

The SDCS framework is a collection of libraries and tools that enable scheduling of applications over available resources in the network. As such, the SDCS is a batch system for a network.

Also, the SDCS enables individual applications to run in parallelized fashion over a number of workstations with minimal change to applications. Due to the loosely-coupled architecture (using a network of independent machines) applications that can be split-up in independently operating parts will benefit maximally from the parallelization functionality in the SDCS.
Parallelization should be at algorithm-level, not at instruction-level. Therefore parallelization is done inside the application, by the application programmer, without special compiler technology or operating system.

Aforementioned functionality makes the SDCS suitable for applications like search-algorithms, numerical computations, cryptographic applications, image-processing, and other compute-intensive applications that can be split up in independent parts.

Benefits of SDCS concept:

* Standard client/server functionality is used such as NFS and UNIX IPC.
* Standard operating system, compilers and development tooling is used. Therefore, development and target environment are the same, and may even be physically the same network.
* Easily scalable to more than 100 workstations, both using Solaris 2.x and SunOS 4.x.
* SDCS works transparent on both uniprocessor systems and multiprocessor systems, and can compensate for different computational strengths.
* Lower costs than traditional supercomputer solutions.

[1] Frans Bouma, a student of the Hogere Informatica Opleiding at Enschede, was involved as a trainee with the implementation of the SDCS from September 1992 until Februari 1993.

The Sun Distributed Computing System (SDCS)

Jos van der Meer (VMX)
Johan van Veen (VMX)
Gertjan van Gent (Sun)
Maarten Westenberg (Sun)
Frans Bouma (VMX) [1]

VMX Professional Services
Sun Microsystems Nederland

1. Introduction

Each user of a Sun network configuration recognizes the consideration that the network as a whole represents a huge calculation capacity. When all that capacity could be used for one application, it's processing could be increased considerably.

It was just this thought that crossed the mind of Sun's Gertjan Van Gent, when he was invited in 1991 by one of Sun's relations to work out a proposal for a configuration with a computation power of minimal 2500 MIPS. Together with Maarten Westenberg of Sun Microsystems Nederland B.V. he implemented a prototype implementation of an execution-environment which could prove the intended speed up that could be realized on about thirty SPARCstations 2.

Based on the proposition that Sun Microsystems Nederland B.V. did for the overall configuration and the demonstration of the prototype, Sun came to terms with the customer. The development of the project software, now known as the Sun Distributed Computing System (SDCS), was contracted out to VMX Professional Services.

The functional specifications for the SDCS software were drawn up by the customer. Though it should be stated, that the customer was much less familiar with the possibilities of a network configuration and the demands that could be made to a programming- and execution-environment than VMX Professional Services. The programming- and execution-environment of the customer had been limited to the standard facilities of a terminal accessed mini-computer under a proprietary operating system. As a consequence VMX Professional Services often advised on the properties of de SDCS that was being implemented .

The SDCS software implements a "simple" concept, but the implementation is of a high quality, because of the strict delimitation of the implemented functionality. Apart from that, the working of the SDCS is easy to understand and both the programming- and the commandline-interface can be learnt quickly. The entire configuration, the SDCS software included, was delivered in December 1991. Since the deliverance no more than some 10 bugs have been detected in the implementation.
The quality of the implementation also proved good during the port to Solaris 2.x. The port went off immaculately and did not show any unknown failings in the code. Apart from that, the result of the port exceeded the expectations of Sun Microsystems and her customer. The current SDCS software transparently supports a mixed configuration of SunOS 4.x and Solaris 2.x.

The SDCS configuration is being used by several programmers. The programmers have absolutely no network programming experience: they alter their (C or Fortran) code as developed for standalone execution, only in a few places and do so by using simple commands from the SDCS-API.

[1] Frans Bouma, a student of the Hogere Informatica Opleiding at Enschede, was involved as a trainee with the implementation of the SDCS from September 1992 until Februari 1993.

2. The premises in the SDCS

The SDCS software was developed to give programmers of algorithms that can be parallellized the possibility to make a non parallellized implementation fit for parallel processing on a network of Sun systems by making some simple changes.
That is all SDCS is, no more and no less.

That is why SDCS software is **not** a distributed operating system. A full implementation of a distributed operating system could *possibly*, but only in combination with a compiler that can parallellize, offer a solution for the problem decribed, but it is obviously "overkill".

Also the SDCS software is **not** a parallel environment. The SDCS software offers the programmer tools to make his application suitable for parallel processing by means of simple actions. How to make the program suitable for parallel processing must be indicated by the **programmer.**

The SDCS software is **not** automatically a solution for all thinkable parallel algorithms. There is a subset of algorithms specifically suitable for processing on the SDCS. We shall describe that subset in Chapter 5, after discussing the way the SDCS works. Namely then, we can use the terminology like it is being used in the SDCS and the reasoning leading to this description will be easier to understand.

The SDCS software **also** implements a batch system on which, whether or not parallell, applications can be processed. This wish was not included in the functional specification of the customer. However the implementation of it makes the application-area of the SDCS much wider and a parallellization of 1 (..) is an extension which fits in very naturally in the SDCS architecture.

3. Properties of the SDCS

Consider the SDCS as the manager of computation power presented by the nodes (Sun computers) in a local area network. Because there are nodes in a network, the SDCS was implemented as a series of processes, having mutual connections and sending messages to each other. It will be clear that a new terminology is necessary to define different components in the configuration and to define the possible actions. Therefor we will first give an overview of the terminology and the different processes.

3.1. The SDCS terminology

Compute points

The user of the SDCS specifies the wanted amount of computation power in an abstract measure, "compute points", an integer. For every machine in the SDCS allowed to be used for computation, the number of compute points representing this machine is known. At the first SDCS installation we have normalized the compute points with respect to the computation power of the SPARCstation 2. To avoid working with multitudes of the computation power of the SPARCstation 2 we have set it’s compute points to 10.

When dividing one calculation over nodes in the SDCS, a node gets a "chunk" of that calculation. That chunk resembles, in size, the contribution the centre delivers to the total number of compute points, allocated for the total calculation.

A thorough estimation of mutual speed differences of the machines will see to it, that all nodes involved have processed their chunk at the same time. A good estimation will give a minimum passage time per assignment.

It will be clear, that the measure of speed for the nodes in compute points does not represent a measure which is exactly appropriate for any individual algorithm. Such a measure does not exist, which will be clarified in the following example.
Suppose we want to determine the primefactors of the numbers 1 to 1000, and the algorithm we use divides the numbers in accordance to the compute points of the machines. Suppose we have 10 identical machines to our disposal, which with certainty represent the same number of compute points; they will still not end their processing at the same time.

For multi-processor machines there are two options. When the applications are unable to use multi-processors (as will be the case with a lacking multi-processor API under Solaris) it is advisable to treat

a 4 processor machine as 4 independant machines.
When the applications do use multi-processors, the machine has to be treated as 1 machine and an overall rating in compute points must be determined.

Compute engines

The compute engines are the working-horses of the SDCS, the nodes where the processing is done. For the SDCS user, all compute engines are mutually exchangeable; the user is only concerned with the number of compute points, not in the way this number is composed for his application.

Thus the application does not use the identity of the machine. Exceptions are possible, like an application using the name of the machine in the name for a temporary file which is stored in a network-wide NFS directory. This is a small extension of regular UNIX conventions; on a single system an application is not interested in the process id under which it works. However it does use it in the naming of temporary files. In the SDCS the combination of the machine name and the process id is unique.

SDCS host server

The SDCS host server is a UNIX proces on a machine, forming a part of the SDCS. Normally this machine is practically reserved for this task, because (as will be shown later) the availability of the host server process must never be endangered. Other tasks the machine, on which the host server process runs, can execute are e.g. NFS server tasks and IP routing.

De SDCS host server maintains a central administration of all the compute engines, the users and their actions. The SDCS host server is also the process to which every request is sent for services of, or information about, the SDCS configuration. When the SDCS host server is stopped, or even has crashed, the entire SDCS configuration has become useless. All calculations on the SDCS are stopped immediately and unconditionally.

The SDCS host server is accessible from SunNet Manager via an agent belonging to the SDCS. To SunNet Manager the whole SDCS configuration is one machine, accessed via the SDCS host server.

It is possible to manage individual machines within the SDCS configuration as ordinary Suns with SunNet Manager. However this does not give any information about the tasks and the use of the machine within the SDCS. Moreover this approach is undesireable, because it brings about unnecessary traffic and what is more important it burdens the machines involved.

SDCS connection server

On every compute engine in the SDCS a connection server is present. The SDCS connection server is a UNIX process which monitors a predefined port on which requests for the compute engine come in. When a compute engine is not connected to a connection server, SDCS considers the machine to be off line.

When a connection server is started, this proces tries to connect to the host server. When the connection is made, the compute engine is marked "on line" by the host server. Then the compute engine can be used for assignments. When the connection cannot be made, the process waits and then the initiative lies with the host server process to check the different compute engines for attainableness.

SDCS application server

The SDCS application server is a UNIX process, started up by a connection server at the moment a request comes in for a specific action. The connection server then immediately returns to its original task.

There are two reasons why a connection server does not handle the application itself:

1. The attainableness of the connection server has to be garanteed because it indicates the attainableness of the compute engine.
2. One way to integrate an *n* processor multi-processor system in the SDCS is to look at it as *n* seperate systems. when this is applied, the connection server has to start up *n* application servers, and therefor be able to start applications on the machine in question.

The main task of the SDCS application server process is to handle the incoming requests. In most of the cases it will be a request to start up the ultimate application, for which the application server acts as a buffer between the SDCS framework and the application.

SDCS client

The client process handles the command line interface of the SDCS. All possible actions on the SDCS can be inititated with the help of the **client** command. Selecting an SDCS service is being done by means of sending along one or more options ("flags"), as is usual in UNIX.

The client process is not responsible for checking the command line, but acts as a buffer between the user and the SDCS framework and thus resembles the application server process.

The client process is always started from the command line, in contradiction to the inter client process which will be discussed later. The client process writes "normal" output on UNIX standard output and error- or feedback-messages on UNIX standard error.

SDCS inter client

This process is started by the client process and is responsible for checking the command line. When the command line was found incorrect, a message is sent back to the client process. When the command line is correct, the inter client process contacts the host server. From the arguments on the command line a request is composed and sent to the host server.
The host server has the possibility to honour the request, refuse the request or put it in a queue.[2] When the request concerns the execution of an application, the inter client receives an overview of machines selected for the execution of the application by the host server. The inter client then sends a message to the selected machines requesting the starting of the application. Both the connection server and the application server are used on the compute engines, as we have explained before.
Two other important tasks of the inter client process are detecting that a compute engine goes down and taking care of a restart of the application started up on the compute engine in question.[3]

An outline of the SDCS processes

The different stages during the execution of an application on the SDCS can be outlined as follows:

client <---> inter client <---> host server

<---> connection server
<---> application server
<---> application server <---> application 0 100 100 0 20 0

<---> connection server
<---> application server
<---> application server <---> application 0 100 100 20 50 1

<---> connection server
<---> application server
<---> application server <---> application 0 100 100 70 30 2

In order to start an application on the SDCS, one uses the client command. This command immediately initiates an inter client process which opens a connection with the host server, when the command line is correct. The inter client submits a request for computation power and when the host server honours the request, the inter client will open a connection with one or more connection servers. These connection servers fork the application server processes, which inherit the connection with the inter client. The

[2] Look at paragraph 3.2.3 for a description of the queue system in the SDCS.

[3] Look at paragraph 3.2.4 for an extensive description of the restart mechanism.

inter client will now send the application name and the corresponding parameters (and additional parameters, see also paragraph 4.2.1) to the application servers. The application servers finally fire up the applications.
What we have described here, does not really differ from a programmatic start of an application on the SDCS. The only difference is the fact that in that case, the inter client process is started by a library function instead of by the client process.

3.2. General properties

In this section we will discuss adaption of applications, dynamical configuration and the queue system. Furthermore a number of facilities for users will be explained. Specific topics for the administrator of the SDCS will be treated in the next section.

3.2.1. Adaption of applications

To be able to use the offered parallel processing facilities one or more small adjustments to the application will have to be made, depending on the kind of application. In order to do so several functions are offered which will reduce adjustments to a minimum. In paragraph 4.2 a number of these functions are discussed. Apart from functions the SDCS offers the following mechanism to limit adjustments to applications.

Every application has to be able to return its results. In the SDCS a special function is available for doing so, but it can be done even more simple. All the output an application presents on UNIX standard output and UNIX standard error, is sent by the application server after the execution. If none of the application-centers intercedes this traffic (see later), it will be delivered at the initiating client by the SDCS framework. This will write a "standard output" message on its UNIX standard output, a "standard error" on its UNIX standard error.

By passing UNIX standard output and UNIX standard error from the initiating client to a file, the similar output of all nodes is collected.

The only change in the output compared to a non parallellized execution is the possible deviation in the sequence of the output. Because the contents of the output for the SDCS framework is unknown (ascii, binair, etc.) SDCS does not change the output. However it is easy for an application-developer to add a good identification per application to the output. Using standard UNIX tools the entire output can then be sorted.

When an application does *not* use the facilities for parallellism, the application can be started **without** adjusting the SDCS. The output of the application as described above is intercepted and sent out.

3.2.2. Dynamical SDCS configuration

An important property of the SDCS is the abillity to configure itself dynamically. Thereby the host server always has a list of the most recent data about the compute engines. As the host server is started up, a file is specified on the command line, in which the names of all machines performing as compute engines are registered. The host server will send these machines a message indicating that the compute engines (read connection/application servers) have to report themselves. This way the host server gets an idea of the compute engines that are "on line" and of those that are "off line".

If a compute engine crashes during the execution of an application, it will be noticed by the inter client process, responsible for executing the assignment. The fact that the connection between the inter client and the application server is being interupted unannounced, gives the inter client an indication of a "crash".

The inter client shall report this problem to the host server and the client process belonging to it. The way in which the application from which a part is now missing will handle this, depends on the options with which the application was started. We shall discuss this later.

When the host server process receives a "crash" message from an inter client, it will then check whether there is really someting wrong with the compute engine. The host server does so the same way as at the beginning, by sending the machine in question a message indicating that the compute engine has to report itself.

When there has really been a "crash", the machine will start a new connection server after booting, which will automatically report itself to the host server.

The file with the names of the machines acting as a compute engine, can be refreshed dynamically. This permits machines to be taken out of the SDCS configuration, to be added to the SDCS configuration or to alter the relation between machines, as is expressed in the compute points.

3.2.3. The queue system of the SDCS

The SDCS has a queue at its disposal in which assignments can be placed. In fact there are three separate queues present on the host server: one with a high priority, one with a medium priority and one with a low priority. Assignments in the queue with a high priority will always be dispatched sooner than assignments in one of the other two queues, regardless of the fact whether assignments in the lower priority queues could be executed. The same relation exists between assignments in the queue with medium priority and assignments in the queue with the lowest priority.

In every queue there is an aim for the highest possible flow. This mechanism is, true enough, not entirely fair, but the choice for this mechanism is justified by the aspiration to use the compute engines to their optimum.

By means of the command line interface it can be indicated in which queue a request has to be placed if it cannot be executed; the default is the queue with medium priority.[4]

3.2.4. Facilities for the users

For the users two kinds of interfaces are offered: a command line interface and an Application Programmers Interface (API). These two interfaces shall be discussed extensively in chapter 4.

In this section we will discuss a number of important SDCS concepts for the user.

SDCS claims and allocations

To be able to carry out an assignment on the SDCS, compute points have to be requested. Hence the user can choose from two options:

1 the direct usage of compute points, the so-called allocation from the "pool";

2 the reservation of compute points, the so-called claiming, followed by allocation from the "claim" that was made.

To the user it is very attractive to claim a number of compute points, so there will always be computing power available for that user. The user can start up several tasks within one claim (at the same time, in batch or after eachother) and he can maintain the claim as long as he wants. This is also the disadvantage of a claim, because the claimed compute points can not be used by anybody else, as long as the claim has not been released. A claim explicitly has to be released by the user (or the administrator).

At creation of a claim, the user sets the name of the claim. This name will later be specified by the user, when machines in the claim have to execute an application.

To execute an application it is necessary to allocate compute points. An allocation can take place from a prior created claim, or from the "pool", which means: all compute engines which are not claimed. At the end of an assignment the allocation is released again. When the allocation took part from a claim, the machines involved are of course still reserved for the owner of the claim.
Allocating from the "pool" is the social way of using the SDCS. The machines involved will then only be occupied during the calculation of the assignment.

For every user allowed to use the SDCS, two limits can be set. One limit indicates the number of compute points each user may claim and the other indicates the number of compute points each user is ultimately allowed to use for execution of assignments. When the claim limit is 0, it means that the user is not allowed to make any claims.

[4] At this instant a maximum priority set per user can not be specified. In the current implementation however this wish was taken into account and the extension will be realized in a following version of the SDCS.

A simple example of using a claim is a reservation, made by a user, of a number of compute points in the morning, that will only be released in the evening . This way the user secures himself of enough compute power to last the whole day.

A more complex example of the use of a claim is the following. Suppose an assignment has to be started up on 20 compute points, and it is known that this assignment will need another 80 compute points during the execution. The user can specify at the start that a claim is needed of 100 compute points and that the assignment is to be started up at 20 compute points.

The SDCS restart

Another facility for users is the so-called restart. Like indicated earlier, the SDCS has the capability to configure dynamically. If the compute engine drops out, the host server will notice it almost immediately. When an application of a user was executed on a compute engine at that moment, the inter client will take care that the task is taken over; this is what we call the SDCS restart.

A "crash" is the sudden disconnection between the application server and the inter client. This is the reason an application can never cause a crash, because when an application drops out, the application server process will take it to be the normal ending of the application. Even a "core dump" of the application is regarded as a normal ending of the application by the application server! The situation is like in UNIX: the applications are protected in such a way that they can not cause a *system crash.*

At such a restart, the work of the compute engine causing the problems, will be started up at another compute engine. This way several "crashes" can be intercepted, essentially until the host server has no more compute engines available. When no more compute engines are available, the host server will wait until one is available again.

The SDCS does not offer facilities to pass partial results of a compute engine to a machine taking over the tasks of the compute engine in case of a crash. This is an application specific problem in which the SDCS can offer little help, because e.g. one compute engine dropping out, can be replaced by several compute engines (each with less compute points).

In practice the absence of the transmission of partial results is not a substantial limitation, because:

1 crashes of compute points are rare;

2 partial tasks are limited in size due to parallellism.

By building the application in such a way that every part transmits the results just before ending, it is (almost) certainly prevented that partial results will appear double in the results of the application as a whole.

Notice that the SDCS configuration initiating the SDCS project is a dedicated configuration. This means that the interest of a restart option is much bigger, in a configuration in which machines have other tasks to carry out apart from the compute engine tasks. When for example desktop workstations are used as compute engine, the user could accidentally turn off the system.

3.3. Properties specifically used for managing the SDCS

Managing the SDCS as a local area network of Sun systems is trivial. When the machines in the SDCS configuration are used for several purposes, additional management of the systems is related to the demands of those applications. To manage machines strictly used for the SDCS, any additional management is not necessary; a standard kernel will do. The machines can be diskless, reboot automatically and, if the rc.local file was adjusted, start up a connection server automatically while booting, etc.

A large SDCS configuration as a whole can in a simple manner represent a "machine" which is comparable in size with a large mainframe. That is why special administration tools have been developed for the SDCS configuration. These facilities will be dealt with in the next section.

3.3.1. The link to SunNet Manager

Together with the SDCS a SunNet Manager agent is delivered, collecting information about the status of the SDCS. The agent collects this information from the host server process, which at any moment has the exact status of all the parts of the configuration.

The link to SunNet Manager was developed from a demand of the first customer; the SDCS software has to provide the opportunity to log and reproduce graphically the degree of occupation of the SDCS configuration. Both facilities (and much more) are offered by SunNet Manager, when an SDCS agent renders the requested information.

3.3.2. An SDCS "quota" system

A user of the SDCS has to be introduced in the SDCS by the SDCS administrator. The administrator accomplishes this by fixing a limit for the amount of computation power the user is allowed to allocate at one time and an amount the user is allowed to have claimed at the same time. These user limits are put in a file which can be passed to the host server. Future extensions of the SDCS, concerned with the user restrictions, also end up in that file. Such as e.g. a fixed maximum priority per user or a limit on the duration of a claim, etc.

3.3.3. An SDCS "accounting" system

The SDCS implements an accounting system for several purposes. The accounting system offers the possiblity to fix a price per compute point per minute for allocation and claim.

This pricelist is determined by the SDCS administrator: he hereby has the possibility to use a different tariff at different times during the day, for different days of the week. That way he can stimulate the SDCS users to use the hours in which the system is less active.

The SDCS administrator can carry through bigger varieties in the pricelist by applying a new pricelist dynamically. From the UNIX' crontab e.g. a pricelist can be made with lower tariffs when a holiday starts.

The obvious purpose of the accounting system is recharging the costs to the users. But accounting can also be used as reporting-mechanism about the degree of occupation and as a mechanism with which deviating usage of the SDCS can be signalled.

3.3.4. An SDCS "logging" system

The SDCS implements a logging system in which, for three categories of events, changes are written out. The three categories are: changes in the SDCS machine configuration (like allocations, claims and crashes), actions by the users (like allocations and claims) and accounting.

The "logging" system has only got a strictly defined output format for accounting. We discourage to write large administration applications using the logging information for the other two categories (at this moment).

4. The SDCS usage

Now that the ideas behind the SDCS and the workings of the SDCS have been described in broad outline, it has become possible to describe the usage of the SDCS. We shall give examples of the use of the SDCS, both from the UNIX command line and from C software.

4.1. SDCS UNIX command line interface

In this paragraph we shall demonstrate the usage of the SDCS from the command line. We make a distinction between executing applications which do use the parallel processing facilities and applications which do not. The examples are executed on a machine named "example", with the prompt "example%".

4.1.1. Parallel processing of an application

In this example we shall first put a claim on 100 compute points of the SDCS configuration. We name this claim "part". The SDCS command we will use is named "client":

example% client -C 100 -I part -K

The meaning of the command line options -C and -I speak for themselves. When the -K option is not specified, the claim is released at the end of the command. The use of this option will become clear later in this paragraph.

We shall use the claim completely to execute an application with 1 application-argument:

example% client -N 100 -I part -A /vol/sdcs/ademo 20

The option "-N" is followed by a number of compute points which have to be allocated. Because the name of the claim is specified on the command line, this allocation occurs within the claim mentioned. The option "-A" is always the final option on the command line and is followed by the application-name and the application-options. The SDCS implementation stops parsing the command-line at the option "-A".

The claim part still exists after this command and we use the same claim for a second application. Again we use the complete size of the claim for the execution. The application *bdemo* has two arguments:

example% client -N 100 -I part -A /vol/sdcs/bdemo 1024 6

Now we are ready and release the claim:

example% client -R part

In the following example an application with 1 argumented called *cdemo* is started. We know that this application starts a second application (*ddemo*), which needs 100 compute points and expects that a claim exists, with the name that has been given as an argument to *cdemo.* The application *cdemo* itself needs only 10 compute points, so altogether we need 110 compute points. We could first apply for a claim, named "part1", then start up the application and finally release the claim again:

example% client -C 110 -I part1 -K
example% client -N 10 -I part1 -A /vol/sdcs/cdemo part1
example% client -R part1

This last example can be made a lot easier by combining the first two lines and leaving the -K flag out:

example% client -C 110 -I part1 -N 10 -A /vol/sdcs/cdemo part1

A request for a claim was has been made under the name "part1" with the size of 110 compute points and the application *cdemo* is started on 10 out of those compute points. At the end of the client commando the claim "part1" is released automatically.

4.1.2. Executing an existing application

The SDCS is equipped with a queue system and can easily be deployed as a batch system. The commands stay the same and existing programs can be executed without change on the SDCS. Because the SDCS is based on compute points, the number of compute engines one can use for an assignment is unknown. To be able to manipulate that, there is a possibility to indicate on the command-line that the wanted number of compute points results in an allocation of a certain number of compute *engines.* The following example illustrates the use of the SDCS as a batch processing system.

An existing application *fdemo* is started via the SDCS. The application does not use the facilities for parallellism, so it is necessary that the application is started up on just one single compute engine. This can be indicated by placing a colon right after the requested number of compute points. After the colon follows the upper limit for the number of compute engines which together form the requested number of compute points, in this example that number is 1.

example% client -N 50:1 -A /vol/sdcs/fdemo 10

Naturally this construction can also be used in combination with claims. When an application does use the facilities for parallellism of the SDCS, the above-mentioned option can be used to command a maximum number of compute engines, e.g. to prevent a loop from being split up in parts that are too small.

The client command is the only SDCS command. Informational requests from users and all administrative operations are also started up with this command.

4.2. Application Programmers Interface

The SDCS API consists of two parts. We need an API with which an application, being executed within the SDCS, can determine which part of the total calculation has to be executed and return the (partial-) results. We call this API the "passive" API.

A second API is needed when resources of the SDCS have requested by, and used from, a C programme. We call this API the "active" API.

4.2.1. The "passive" SDCS API

Extra parameters within the SDCS

An application that is executed within the SDCS receives six (6) extra parameters from the SDCS:

1. The offset in compute points of the current run.
2. The size in compute points of the current run.
3. The total number of compute points the current run executes.
4. The offset in compute points of the current process.
5. The rating in compute points of the current process.
6. The offset in compute engines (!) of the current process.

An example can clarify this:

example% client -N 100 -I part -A /vol/sdcs/bdemo 1024 6

Suppose the host server places four (4) compute engines to the disposal of the calculation with the next compute points: 20 20 30 and 30. The SDCS framework now starts up the same /vol/sdcs/bdemo on these compute engines. The four executions only differ in their parameters.

Succesively the four executions of /vol/sdcs/bdemo are:

a. 0 100 100 0 20 0

b. 0 100 100 20 20 1

c. 0 100 100 40 30 2

d. 0 100 100 70 30 3

The third parameter seems redundant, because it is the difference of the second and the first parameter. Using the current implementation of the automatic restart that is indeed always the case, but in the first implementation of the automatic restart it did not work that way. For backwards compatibility the third parameter is still available.

The automatic restart also sees to it that the final parameter stays a unique running number within the current process. Suppose the third compute engine drops out. At a restart the host server is then requested to replace 30 compute points. It may occur that the host server at that moment offers this replacement in the form of two compute engines of respectively 25 and 15 compute points. These two new executions of /vol/sdcs/bdemo succesively get the following parameters:

c*. 0 100 100 40 25 2R0

c**. 0 100 100 65 5 2R1

We see a new running number for the two executions, which shows that the third (!) engine was concerned in the original execution. Within the restart we get a numbering starting at 0 again, this new numbering shows after the capital R in the final parameter. Further more another phenomenon appears: the compute engine of 15 compute points is treated as if it has 30 - 25 = 5 compute points. For that reason it shall process less work as would be obvious regarding its relative power and it is finished sooner. In the implementation of the SDCS we assume that restart is an exception and therefor a simple implementation is preferable. Thus we do not exert ourselves in the implementation to minimize the processing time at this point.

A compute engine functioning as a replacement can also drop out. When the first replacement drops out (of 25 compute points) and is replaced by another of 35 compute points, the parameters of this new execution of /vol/sdcs/bdemo will be:

c***. 0 100 100 40 25 2R0R0

The 6 SDCS parameters are passed from argv[1] up to and including argv[6] to the executable application, when the application was started up with the option -A.
When the option -E was used instead of -A the 6 parameters will be offered to the application by means of an environment variable .

Loop arguments from frame arguments

To get a correct and efficient division of calculation that is to be executed it is important to interpret the extra SDCS arguments correctly.

The function "loop_from_frame_args" divides a linear stretch with a starting and end point that must be given, by means of the SDCS parameters. An example of it's use is:

```
result = loop_from_frame_args(0, loop_size,
            &my_loop_start, &my_loop_end, argc, argv);
```

Because every execution of the application calls the same loop_from_frame_args, the total loop from 0 to *loop_size* is correctly and efficiently divided over the participating compute engines.

The source code from loop_from_frame_args is delivered with the SDCS software as a basis, for functions that must divide the executable calculations in other contexts, e.g. the division of gridpoints in an *n* dimensional cube.

4.2.2. The "active" SDCS API

The "active" SDCS API enables:

1. Applying for and using SDCS resources from an executable.
2. Intercepting and filtering the SDCS messages-traffic from an executable.
3. Sending messages to nodes in an execution.

It will be clear that with the help of this API the SDCS can execute a much more complex type of application. An example of the type of application for which such an API is needed, is a heuristic finding algorithm, in which a branche has to be readjusted (restrained, speeded up, broadened) based on the results of the other branches.

In this article we shall only deal with making requests and using the SDCS resources from a C program, in the context of the first SDCS installation this was the complete demanded functionality.

The programmatical interface of the SDCS client

The power of the SDCS lies in the simplicity of the usage. That is why the "active" SDCS API is made up of a 1 to 1 representation of the SDCS command line interface. When a new (or a partial-) application has to be started up from a C programme, it can be done like this:

```
clientfd = clientl("-N", "100", "-I", "part", "-A",
                   "/vol/sdcs/bdemo", "1024", "6", NULL);
```

5. Applicability of the SDCS concept

In the SDCS implementation we have strictly seperated application aspects and parallel aspects. The SDCS mainly consists of processes composing the SDCS. Only a very small part is formed by a library.

Strictly spoken, as the SDCS implementors, we can say very little about which application is and which is not appropriate for execution on the SDCS. The application experts are the ones who should judge this.

Still there are a number of criteria making an application highly suitable for execution on the SDCS. Divergences from these criteria do not on forehand make the SDCS concept unsuitable, but cause that more changes in the application are necessary for execution on the SDCS.

1. The application has to be seperable into parts. The number of parts has to be in order of magnitude of the number of participating compute engines of the SDCS.

The limits of this criterion are vague: on an SDCS with 100 compute engines an application that has to be split up in 10.000 parts is still executable. In the opposite direction, a similar remark can be made: the SDCS is highly suitable for batch processing, which can be compared to applications that can only be seperated into one part.

2. The mutual dependence between the parts has to be limited.

As we have noticed, there is a programmatic interface with which the parts of an application on the compute engines can mutually exchange information (via their parent in the execution hierarchy). This mechanism is very powerful, but insufficient when in a normal implementation the parts frequently use eachothers data structures.

3. The execution-time of application parts has to be in the order of magnitude of tens of seconds.

This measure is also a directive. The overhead the SDCS places on the remote execution of an application part on a compute engine, is almost constant and amounts to a few seconds. So even when each application part is equal in size, and takes only a few seconds, an execution on tens or hundreds of compute engines can be a significant speed up.

However when each application part has a size that has to be expressed in milliseconds, it is unlikely that the SDCS can signify a speed up.

6. Availability of the SDCS, supported platforms

The SDCS is currently supported on SPARC. The configuration can be composed of all types of Sun-4, SPARCservers and SPARCstations, including multi-processor machines.

The SDCS is available for both Solaris 1.x and Solaris 2.x. Besides, the SDCS supports mixed configurations of SPARC based machines under Solaris 1.x and Solaris 2.x. On this kind of configuration only those applications can be executed that are tested under Solaris 2.x binary compatibility mode.

There is no fundamental reason why the SDCS should not be suitable for other platforms than SPARC/Solaris. A heterogeneous configuration can also be supported by the SDCS, when the internal communication within the SDCS is extended with a universal data representation (like XDR).

At this instant however, there are no concrete plans to port the SDCS to other platforms.

7. Future extensions to the SDCS

The SDCS software was developed and is being maintained and extended by a small team of software engineers. Therefor the simplicity and formal correctness of the internal implementation is crucial.

Three aspects play a role when considering what functionality should be integrated into the SDCS:

1. Are the extensions demanded by a potential SDCS buyer ?
2. Are the requested extensions in agreement with the design and architecture of the SDCS ?
3. Is the potential buyer of the SDCS willing to finance the implementation of the adjustments suggested by him?

The current SDCS was developed this way, based on a much simpler version developed on order of the first customer. In this way e.g. the priority-queue and the accounting mechanism were added.

Moreover the internal implementation of the actual SDCS has improved a great deal compared to the first implementation: the extensibility of the current code has increased quite a bit.

There is one adjustment the software engineers of the SDCS would like to realize themselves, which is in the field of the division of the application on compute engines and the rating in compute points. We think that a different approach, the one we have in mind, could lead to an application specific, dynamic speedrating for this problem and thus lead to a shortening of the processing time of individual assignments.

8. Conclusions

In this paragraph the most important properties of the SDCS will be enumerated. These properties are divided into two parts, the first part concerns general properties, of importance particularly for the user, the second part concerns properties specially for the administrator.

Properties of the SDCS

- The SDCS can be looked at as an administrator of computation power, by clustering a number of random Sun werkstations/servers. The software was developed to enable programmers of algorithms to make a non parallel implementation suitable for parallel processing, by making a few simple adjustments.
- The SDCS software also implements a batch system on which, whether or not parallel applications, can be processed. Therefor the SDCS is equiped with a priority queue system.
- The output of applications on UNIX standard output and UNIX standard error is intercepted and passed on to a location from where the assignment for execution of the application was given.
- Existing applications not using the parallel facilities of the SDCS can be started up on the SDCS **without** adjustments. To be able to use the parallel processing capabilities, one or more small changes have to be made to an application, depending on the type of application.
- The SDCS can configure itself **dynamically.** An important part is the **restart** mechanism, guaranteeing the continuity of the application that was started up.
- The SDCS offers the users a simple command line interface and an Application Programmers Interface (API) which can be used from both C and FORTRAN. The API was made up out of a 1 to 1 representation of the command line interface and is therefor very easy to use.

Properties specially for administrators

- To manage machines solely used for the SDCS, no additional administration is necessary; the machines use a standard kernel, they can be diskless, boot automatically and, if the rc.local file was adjusted, can start up a connection server automatically while booting, etc.
- Together with the SDCS a so-called *proxy agent* is delivered, which enables the mapping of the situation of the SDCS with the help of SunNet Manager.
- The SDCS has a *quota*, *accounting* and *logging* system at its disposal with which the administrator can check the level of occupation of the SDCS.
- The SDCS is available for both Solaris 1.x and Solaris 2.x. and what's more, the SDCS supports mixed configurations of SPARC based machines under Solaris 1.x and Solaris 2.x. On this type of configuration only those applications can be executed that are tested under Solaris 2.x binary compatibility mode.

9. Acknowledgements

We thank Gertjan van Gent, Maarten Westenberg, Raymond Groebbé and Roger Kellerman from Sun Microsystems Nederland B.V. Without their understanding and efforts the SDCS project would not even have existed.

10. References

For more information about the SDCS you can contact:

VMX Professional Services
Computerweg 1
3821 AA Amersfoort
Nederland

Tel: (31) 33 560254
Email: sdcs_info@vmx.nl