Руководство пользователя системы метакомпьютинга X-Com^2

Версия системы от 16.03.2010.

Содержание

0. Быстрый запуск и освоение системы X-Com^2

Для быстрой установки, настройки, запуска и освоения системы X-Com^2 в среде ОС Linux (для MS Windows отличаются только способы запуска программ) выполните следующие шаги:

1. Скачайте последнюю версию дистрибутива системы со страницы http://x-com.parallel.ru/download.html

2. Распакуйте полученный архив

3. Откройте консоль и перейдите в каталог `xcom/gserv`

4. Запустите команду `./Xserv.pl power.ini`

   Если после запуска этой команды будут получены сообщения Perl об отсутствии установленных модулей, выполните установку этих модулей согласно руководству к используемой версии Perl.

5. Откройте еще одну консоль и перейдите в ней в каталог `xcom/gcli`

6. Запустите команду `./client.pl –s http://localhost:65002`

Дальнейшие шаги:

1. Настройте сервер X-Com на прием соединений от клиентов из сети Интернет. Для этого в файле power.ini вместо localhost укажите доменное имя компьютера или его IP-адрес

2. Запустите несколько клиентов, одновременно работающих с одним сервером, на одном или на разных компьютерах, изменив соответственно адрес сервера при запуске

3. Поэкспериментируйте с аргументом (входным параметром) задачи Power: в файле power.ini в строке taskArgs вместо значения "2" укажите другое число.

4. Поэкспериментируйте с серверной частью задачи (файл xcom/tasks/Power.pm):

   - увеличьте количество порций с 50 до 1000 (для этого измените значение, возвращаемое функцией getLastPortionNumber),
   - измените содержимое порций (например, изменив коэффициент умножения в функции getPortion),
   - измените функцию addPortion таким образом, чтобы результаты вычислений сохранялись в файл с заданным именем.

5. Поэкспериментируйте с клиентской частью задачи (файл xcom/tasks/gctask):
   - измените алгоритм обработки порции, например, заменив возведение в степень другой арифметической операцией.

После изменения клиентской части не забудьте обновить и архивы с задачей для узлов!

1. Архитектура и основные возможности системы X-Com^2

Система X-Com^2 предназначена для организации распределенных вычислительных сред и проведения вычислений в таких средах. Общая архитектура системы X-Com^2 приведена на рис.1.

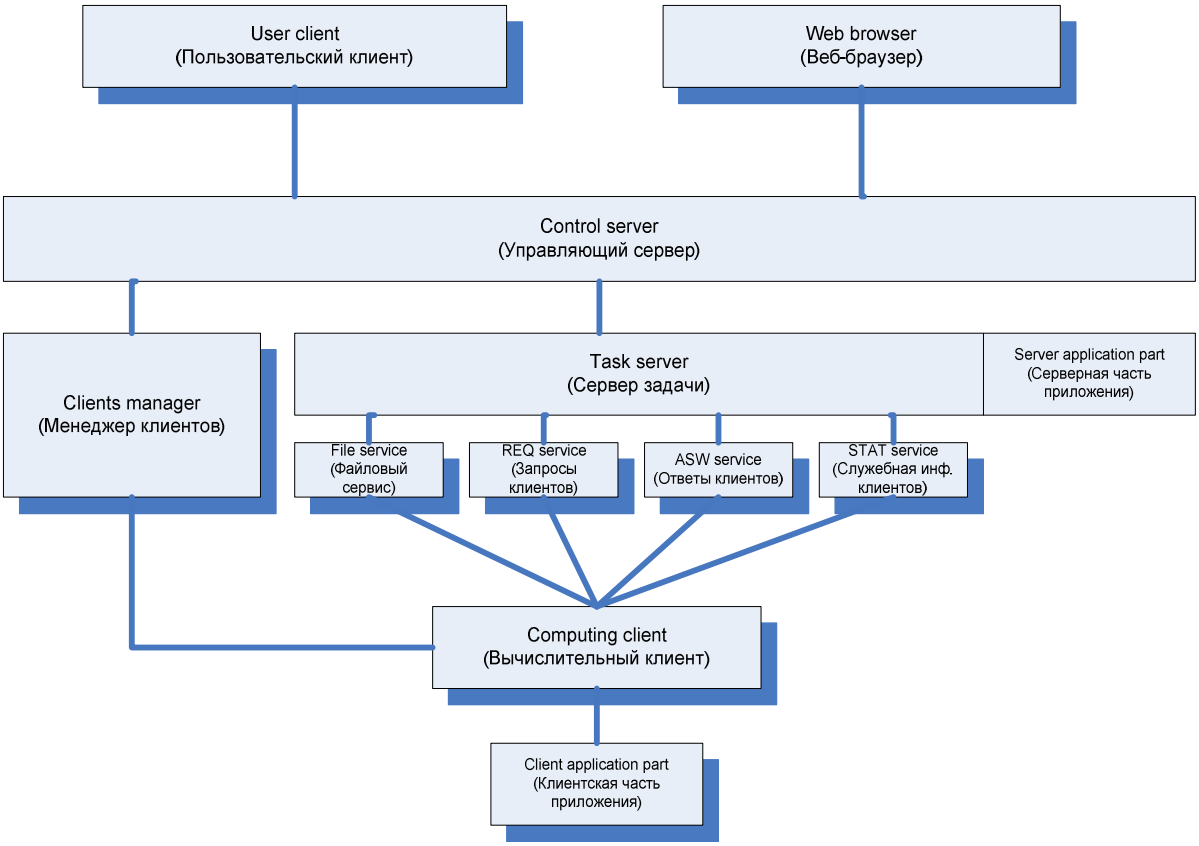

Рис. 1. Архитектура системы X-Com^2

Прямоугольники на рисунке соответствуют процессам системы, причем каждый такой процесс может выполняться на отдельной физической машине. Общение между процессами осуществляется в рамках протокола TCP/IP.

В архитектуре выделяются три уровня – пользовательский уровень, серверный уровень и уровень вычислительных узлов.

Процессы пользовательского уровня:

- Пользовательский клиент – компонент системы, обеспечивающий интерфейс между пользователями и системой метакомпьютинга. С его помощью пользователь может поставить задачу в очередь на выполнение и отслеживать ее состояние. При постановке задания в очередь пользователь формирует файл описания задания, в котором указываются входные данные прикладной задачи и требования к ее запуску.
- Веб-браузер – обычный интернет-браузер, с помощью которого пользователь может следить за ходом расчетов и состоянием распределенной среды в наглядной форме. При наличии веб-интерфейса к прикладной задаче с помощью браузера также может осуществляться постановка задачи в очередь.

Процессы серверного уровня:

- Сервер задачи – компонент, обеспечивающий выполнение одной конкретной прикладной задачи. Он взаимодействует с серверной частью прикладной задачи,

которая генерирует порции данных и осуществляет финальную обработку результатов, приходящих от клиентов с вычислительных узлов. Сервер задачи реализует логику выдачи порций клиентам. С клиентами на вычислительных узлах сервер задачи взаимодействует не напрямую, а через процессы-сервисы, обозначенные на рисунке как File service, REQ, ASW и STAT. Файловый сервис – это фактически файловый сервер, у которого клиент запрашивает файлы прикладной задачи и вспомогательные файлы, если они необходимы. Через сервис REQ (request) клиент запрашивает очередную порцию данных у сервера задачи. Результаты выполнения прикладной задачи над данными очередной порции клиент пересылает сервису ASW (answer). Сервис STAT (statistics) обеспечивает обмен контрольной информацией между клиентами и серверной частью. Выделение сервисов REQ, ASW и других отельными процессами обусловлено требованием распределения нагрузки между компонентами системы. В этом случае сервисы берут на себя всю предварительную обработку входящих и исходящих порций. В качестве файлового сервера может применяться любой http-сервер.

- Управляющий сервер – компонент, обеспечивающий управление прохождением заданий в распределенной среде, в частности, реализующий логику очередей заданий или же логику одновременного выполнения задач. На каждое запускаемое задание управляющий сервер порождает свой сервер задачи. Управляющий сервер транслирует команды – например, на срочное прекращение расчета – серверу задачи, которые затем передаются клиентам, работающим с данным сервером.
- Менеджер клиентов – компонент, к которому все клиенты обращаются при первом запуске. Менеджер клиентов перенаправляет вычислительные узлы к тем серверам задач, требованиям которых они соответствуют. По завершению работы с заданием клиенты вновь обращаются к менеджеру.

На уровне вычислительных узлов функционирует клиент X-Com^2, который получает от сервера задач очередную порцию данных, запускает клиентскую (вычислительную) часть прикладной задачи и передает результаты вычислений обратно на сервер.

Важно отметить, что сервер задач в совокупностью с сервисами обработки клиентских запросов (REQ, ASW и т.д.) реализован как самостоятельный программный модуль, не требующий для своей работы управляющего сервера и менеджера клиентов. В простейшем случае эти компоненты могут отсутствовать, запуск сервера задач может производиться вручную, а клиенты общаться с сервером задач без посредства менеджера. Инфраструктура управляющий сервер-менеджер клиентов предназначена для управлением потоками заданий в распределенной среде, и в частности, для организации очереди заданий.

В дальнейшем будем придерживаться следующей терминологии:

- **серверная часть X-Com^2, сервер X-Com^2** – сервер задач совместно с сервисами обработки клиентских запросов;
- **клиентская часть X-Com^2, клиент X-Com^2** – вычислительный клиент;
- **подсистема управления заданиями** – управляющий сервер, менеджер клиентов и пользовательский клиент.

2. Программные и аппаратные требования компонентов системы X-Com^2

Все программные компоненты системы X-Com^2 реализованы на языке Perl. Рекомендуется использоваться Perl не ниже версии 5.8.

Серверная часть системы требует для работы следующие модули Perl:

- libwww-perl (LWP)
- Time::HiRes
- XML::Parser
- XML::Writer

Клиентская часть системы использует модули:

- libwww-perl (LWP)
- Archive::Tar
- XML::Parser
- XML::Writer

Подсистема управления заданиями использует модули:

- libwww-perl (LWP)
- XML::Parser
- Win32::Process (только при работе под MS Windows)

Перечисленные модули доступны в архиве Comprehensive Perl Archive Network (CPAN, http://www.cpan.org/) и могут быть установлены стандартными средствами Perl (в зависимости от платформы и дистрибутива). Остальные используемые модули обычно включаются в поставку по умолчанию и должны быть установлены вместе с интерпретатором Perl.

Для работы системы X-Com^2 в среде MS Windows рекомендуется использовать дистрибутив Strawberry Perl (http://strawberryperl.com/). Для установки дополнительных модулей в Strawberry Perl достаточно в командной строке вызвать команду

```
cpan <имя_модуля>
```

и ответить на несколько вопросов программы установки.

Клиентская часть прикладной упаковывается в архив типа .tar.gz. Утилиты tar и gzip в ОС Linux доступны по умолчанию, для MS Windows необходимо использовать их аналоги (например, доступные в рамках проекта GnuWin32 – http://gnuwin32.sourceforge.net/). Также можно воспользоваться возможностями архиватора 7zip (http://www.7-zip.org/).

3. Состав дистрибутива системы X-Com^2

Архив с дистрибутивом системы X-Com^2 имеет следующую структуру (перечислены только основные файлы и каталоги):

xcom/gcli	Клиентская часть системы
xcom/gserv xcom/gserv/logs	Серверная часть системы Каталог для хранения лог-файлов сервера
xcom/proxy	Временный каталог для хранения порций, буферизованных промежуточным сервером
xcom/tasks xcom/tasks/Power xcom/tasks/TestPl	Каталог для прикладных задач Демонстрационная задача Power Тестовая задача TestPl
xcom/xqserv xcom/xqserv/tasks	Подсистема управления заданиями Каталог для хранения файлов настроек сервера, создаваемых подсистемой управления заданиями
xcom/utils	Различные вспомогательные утилиты
xcom/ssl	Утилиты для генерации и работы с сертификатами

Файлы с программным кодом, реализующие основные компоненты архитектуры системы (см. Рис. 1):

- управляющий сервер, менеджер клиентов:
  - xcom/xqserv/XQServ.pl
  - xcom/xqserv/XQUtils.pm – вспомогательные функции
  - xcom/xqserv/start_xcom.sh – необходим для запуска серверов X-Com^2
- пользовательский клиент:
  - xcom/xqserv/xqc.pl (также требует для работы XQUtils.pm)
- сервер задачи:
  - xcom/gserv/Xserv.pl – запуск процессов сервера X-Com^2, разбор файла настройки
  - xcom/gserv/XservProcessor.pm – логика сервера задачи
  - xcom/gserv/XservProxy.pm – логика промежуточного сервера
  - xcom/gserv/XServAPI.pm – реализация API сервера
  - xcom/gserv/XservUtils.pm – вспомогательные функции
- сервисы File, REQ, ASW, STAT:
  - xcom/gserv/XservSubserver.pm (также требует для работы XservUtils.pm)
- вычислительный клиент:
  - xcom/gcli/client.pl – запуск клиента
  - xcom/gcli/* – вспомогательные модули

4. Адаптация прикладных программ

4.1. Общие сведения об адаптации

Адаптация прикладной задачи к распределенным вычислениям с использованием системы метакомпьютинга X-Com^2 подразумевает, что задача может быть разбита на множество независимых блоков – вычислительных порций. Серверная часть прикладной задачи генерирует входные порции данных и передает их (посредством транспорта системы метакомпьютинга) вычислительным клиентам на узлах. Клиенты на узлах запускают клиентскую часть прикладной задачи, которая производит необходимые вычисления. Результаты вычислений передаются серверной части.

Крайне желательно формировать вычислительные порции таким образом, чтобы на среднем по характеристикам вычислительном узле их время обработки составляло от минуты до часа.

4.2. Серверная часть прикладной программы

В системе X-Com^2 заложено два интерфейса взаимодействия серверной части с прикладной программой:

1. Files API. Этот способ взаимодействия X-Com^2 с прикладной программой подразумевает размещение исходных порций на файловой системе центрального сервера X-Com. Результаты расчета также складываются в файловую систему основного сервера.

2. Perl API. Используется в тех случаях, когда входные порции не являются файлами, либо когда для создания порций необходима дополнительная логика. Интерфейс подразумевает написание модуля – серверной части задачи на языке Perl, реализующего заданный набор функций. Механизм также позволяет (средствами Perl) подключать библиотеки, написанные на других языках программирования, например, C/C++.

4.2.1. Интерфейс Files API

Данный интерфейс не предусматривает какого-либо программирования серверной части задачи. Каждая порция данных соответствует одному файлу, результаты обработки порций также сохраняются в файлах. В качестве параметров задачи при запуске сервера задач указывается:

- путь к входным файлам;
- файл со списком входных файлов с опциональными аргументами;
- каталог, в который будут сохраняться файлы с результатами
- расширение имени файла с результатом; результирующий файл будет именоваться как <имя_входного_файла><расширение>

Подробную информацию об этих параметрах см. далее в разделе "Запуск серверной части X-Com^2".

Файл со списком входных файлов должен иметь следующую структуру

```
<имя_файла_1> [аргумент_1_1] [аргумент_1_2] […]
<имя_файла_2> [аргумент_2_1] [аргумент_2_2] […]
…
<имя_файла_N> [аргумент_N_1] [аргумент_N_2] […]
```

- т.е. в начале каждой строке по одному имени файла, после имени файла через пробел могут располагаться опциональные параметры, которые будут переданы клиентской части программы.

Первым параметром в клиентскую часть задачи всегда передается имя файла, из которого была взята порция. Это может быть полезно в тех случаях, когда имя файла существенно для обработки на клиенте.

Пример содержимого файла списка:

```
file01.txt 10 100
file02.txt 20 200
file03.txt 40 400
```

Здесь file01.txt, file02.txt, file03.txt – имена файлов, содержимое которых будет передаваться клиентам в качестве порций, а числа после имен ("10 100", "20 200", "40 400") – опциональные параметры прикладной задачи, которые будут использованы для обработки этих порций. Если в качестве общих для задачи аргументов (параметр taskArg в файле настройки сервера, см. далее) были заданы параметры "-a –b", то реально в задачу параметры буду переданы в виде строк "file01.txt –a –b 10 100", "file02.txt –a –b 20 200", "file03.txt –a –b 40 400", т.е. сначала имя файла, затем общие аргументы, затем опциональные аргументы. Общие и/или опциональные аргументы могут отсутствовать.

4.2.2. Интерфейс Perl API

Данный интерфейс подразумевает написание библиотеки-модуля на языке Perl, реализующей следующие функции:

initialize(taskArg)	Функция инициализации серверной части прикладной программы. На вход подаются аргументы, определяемые параметром запуска taskArg
getFirstPortionNumber()	Данная функция возвращает номер первой порции (это может быть любое целое число). Обычно порции нумеруются с единицы.
getLastPortionNumber()	Данная функции возвращает номер последней порции (любое целое число, большее, либо равное номеру первой порции). Если точно число порции не известно, данная функция должна вернуть ноль.
isFinished()	В случае если точное число порций неизвестно, данная функция определяет, закончены ли вычисления. Она вызывается каждый раз после запроса очередной порции. Если число вычислительных порций точно известно и определено в предыдущих двух функциях, isFinished() должна всегда возвращать ноль. В противном случае функция должны возвращать 0, пока вычисления еще не закончены, и 1, когда расчет должен завершиться.
getPortion(N)	Эта функция возвращает порцию с номером N
addPortion(N,data)	Функция обработки результата расчета порции данных номер N. В нее передается номер готовой порции данных и строка с результатами вычислений.
finalize()	Функция, которая будет вызвана после завершения расчета. Единственная необязательная функция.

Все функции, кроме finalize(), должны быть обязательно реализованы в модуле.

Пример реализации серверной части простейшей прикладной программы Power, которая генерирует 50 вычислительных порций, пронумерованных от 1 до 50, а телом вычислительной порции является удвоенное значение ее номера:

```
package Power;                # Имя пакета = имя задачи

sub initialize {
  my $taskArg = $_[0];        # считываем аргументы задачи
  print STDERR "Power: initialization argument is $taskArg\n";
}

sub getFirstPortionNumber {
  return 1;                   # номер первой порции = 1
}

sub getLastPortionNumber {
  return 50;                  # номер первой порции = 50
}

sub isFinished {
  return 0;                   # число порций задано, поэтому возвращаем 0
}

sub getPortion {
  my $N = $_[0];              # получаем номер порции в $N
  my $prt = $N*2;             # формируем содержимое порции ($N*2)
  print STDERR "Power: portion $N created, value is '$prt'\n";
  return $prt;                # отдаем порцию
}

sub addPortion {
  my ($N, $data) = ($_[0], $_[1]); # получаем номер порции и результат
  print STDERR "Power: portion $N processed, result is '$data'\n";
}

sub finalize {
  print STDERR "Power is finished.\n";    # завершение расчета
}

1;               # эта единица в конце файла должна быть обязательно
```

Для быстрой адаптации серверной части задачи, написанной для системы X-Com первого поколения (т.е. в виде библиотеки на языке C/C++), к текущей версии системы можно использовать ПО SWIG (http://www.swig.org/) – средство связи кода на C/C++ со скриптовыми языками. В качестве описания интерфейса библиотеки для SWIG можно использовать следующий шаблон (файл OldProgram.i):

```
%module OldProgram
%{
extern int initialize(char *arg);
extern long getFirstPortionNumber();
extern long getLastPortionNumber();
extern int isFinished();
extern char *getPortion(long n);
extern void addPortion(long n, char *data);
%}

extern int initialize(char *arg);
extern long getFirstPortionNumber();
```

```
extern long getLastPortionNumber();
extern int isFinished();
extern char *getPortion(long n);
extern void addPortion(long n, char *data);
```

В результате работы по исходному коду серверной части и представленному описанию SWIG сгенерирует модуль для Perl (файл в формате .pm) и преобразованный код на C, который необходимо будет откомпилировать совместно с библиотеками используемого дистрибутива Perl. Полученный бинарный файл библиотеки функций (с расширением .dll для ОС Windows и .so для ОС Linux) необходимо разместить в каталоге xcom/gserv. Пример последовательности команд для ОС Windows (с использованием Strawberry Perl, в состав которого входит компилятор gcc):

```
swig.exe –perl5 OldProgram.i
gcc –c OldProgram.c OldProgram _wrap.c –I C:\strawberry\perl\lib\CORE\
gcc –shared OldProgram.o OldProgram_wrap.o
      C:\strawberry\perl\lib\CORE\libperl510.a –o OldProgram.dll
```

Подробная информация о работе со SWIG находится в соответствующем руководстве.

4.3. Клиентская часть прикладной программы

4.3.1. Общие сведения о клиентской части задачи

Основу клиентской части задачи, как правило, составляет вычислительный модуль прикладной программы – обычное приложение, состоящие из произвольного числа исполняемых файлов и файлов данных, с произвольной структурой каталогов. Для взаимодействия между вычислительным модулем и системой X-Com[2] в ней реализована поддержка двух интерфейсов взаимодействия:

- Perl – интерфейс, использующийся по умолчанию, полностью аналогичный интерфейсу с использованием файла gctask в системе X-Com предыдущего поколения;
- Processes – интерфейс, не требующий программирования клиентской части задачи (реализаций функций в файле gctask). Вместо этого в параметрах настройки сервера указываются команды для инициализации и обработки порций, а также имена входных и выходных файлов, в которые записывается порции и откуда считываются результаты ее обработки.

Независимо от выбора интерфейса, все файлы, необходимые для работы клиентской части задачи, запаковываются в архив типа .tar.gz (например, командой tar cvzf). Имя файла строится по следующему принципу:

```
<имя_задачи>-<операционная_система>-<процессорная_архитектура>.tar.gz
```

Имя задачи может быть любым. В настоящее время поддерживаются следующие предопределенные значения для программно-аппаратных платформ, определяемых и поддерживаемых клиентом:
Операционные системы:

- linux – ОС семейства Linux
- MSWin32 – ОС семейства Microsoft Windows

Процессорные архитектуры:

- x86 – стандартные 32-битные процессоры Intel-совместимой архитектуры
- x86_64 – 64-битные Intel-совместимые процессоры

- ia64 – архитектура Intel Itanium

Например, файлы клиентской части для задачи с именем Power будут иметь следующие имена для соответствующих архитектур:

- Power-linux-ia64.tar.gz
- Power-linux-x86_64.tar.gz
- Power-linux-x86.tar.gz
- Power-MSWin32-x86.tar.gz

Файлы с клиентскими частями для всех программно-аппаратных платформ размещаются в каталоге xcom/tasks/<имя_задачи> . Т.е. для предыдущего примера файлы размещаются так:

- xcom/tasks/Power/Power-linux-ia64.tar.gz
- xcom/tasks/Power/Power-linux-x86_64.tar.gz

и т.д.

В этом же каталоге размещаются вспомогательные файлы, которые автоматически скачиваются (по HTTP) клиентами в рабочую директорию. Имена файлов (без пути) перечисляются в параметре dlFiles серверной части (см. далее).

4.3.2. Интерфейс Perl

Для обмена данными с вычислительным модулем задачи используется специальный интерфейс, размещенный в файле с именем gctask, который должен находится на первом уровне в архиве с клиентской частью задачи. Этот файл представляет собой perl-скрипт, в котором должны быть определены две функции:

gcprepare	Данная функция предназначена для инициализации вычислительного модуля прикладной программы. Функция должна возвращать единицу, если инициализация прошла успешно и ноль, если в процессе инициализации произошли ошибки. Если при вызове клиента были заданы дополнительные аргументы с помощью параметра –add-arg, эти аргументы будут переданы данной функции как параметры.
gctask	Данная функция подготавливает и запускает вычислительную часть прикладной программы. Функция имеет 5 обязательных аргументов: $task имя задачи, которая сейчас решается $taskarg аргумент для текущей порции данных $portion номер текущей порции данных $din имя файла с исходными данными (файл находится в текущем каталоге) $dout имя файла, в который надо записать результат обработки данной порции (файл также должен находится в текущем каталоге) $addarg (необязательный параметр) аргументы, передаваемые клиентской части через параметр –add-arg. Функция должна возвращать единицу, если расчет произведен успешно и ноль, если в процессе расчета произошла ошибка.

Пример реализации простейшей клиентской части прикладной задачи Power. Задача предполагает, что в качестве порции ей передается целое число, а в качестве результата отдается результат возведения этого числа в степень, передаваемую в качестве аргумента задачи (в параметре taskArg). Вследствие простоты задачи вся клиентская часть состоит из единственного файла gctask (вычислительный модуль отсутствует):

```
sub gcprepare {
  return 1;                          # подготовка завершена - возвращаем 1
}

sub gctask {
  my ($task, $taskarg, $portion, $din, $dout) = @_;  # входные параметры
  open IN, "< $din";                 # открываем файл $din с входной порцией
  my $n = <IN>;                      # читаем число в переменную $n
  close IN;                          # закрываем файл
  my $res = $n ** $taskarg;          # возводим в степень
  open OUT, "> $dout";               # открываем файл для записи вых. данных
  print OUT $res;                    # пишем результат
  close OUT;                         # закрываем файл
  return 1;                          # успешно - возвращаем 1
}

1;                                   # эта единица должна быть обязательно
```

Файл gctask вместе с исполнимым модулем задачи (и другими файлами, требуемыми для работы программы на узле) запаковывается в архив .tar.gz клиентской части задачи.

4.3.3. Интерфейс Processes

См. описание параметров сервера xCliAPI и др. в разделе 5 настоящего руководства.

5. Запуск серверной части X-Com^2

Серверная часть системы X-Com^2 вместе со своими вспомогательными модулями располагается в каталоге xcom/gserv.

Серверная часть системы может работать в нескольких режимах:

- Основной сервер (обычный, базовый режим). Серверная часть настраивается непосредственно на задачу, указанную в файле конфигурации, с использованием API Perl или Files.
- Промежуточный сервер. В этом случае в настройках сервера указывается адрес вышестоящего сервера. Промежуточный сервер буферизует запросы между вышестоящим сервером и нижележащими узлами, фактически выполняя функции прокси.
- Сервер запросов. Работает только компонент системы, отвечающий за разбор клиентских HTTP/XML запросов (сервис запросов). Этот компонент обменивается данными с сервером задачи, который может находиться на другом физическом сервере. Такой режим применяется в том случае, если значительная часть нагрузки на серверную часть приходится на разбор или создание непосредственно запросов.

В рамках одной среды возможна совместная работа серверов X-Com^2, запущенных в различных режимах.

При запуске серверной части изначально порождаются 2 процесса: сервер задач и сервис запросов (сервис обмена данным с клиентами системы). Между собой эти два процесса общаются либо через TCP/IP (в этом случае процессы могут быть разнесены по разным физическим машинам), либо через сокеты UNIX (тогда процессы должны работать на одной машине). Первый способ общения более универсальный, второй демонстрирует более высокую производительность серверной части при интенсивном потоке запросов, однако может применяться только на машинах, работающих под управлением ОС UNIX/Linux.

Для конфигурирования и запуска серверной части системы используются файлы конфигурации, имя которых указывается в качестве единственного параметра командной строки. Запуск серверной части командой (текущим каталогом должен являться каталог xcom/gserv):

```
./Xserv.pl <config.ini>
```

в ОС Linux, или

```
perl Xserv.pl <config.ini>
```

где config.ini – имя конфигурационного файла, определяющего режим работы серверной части. Строки файлов настройки представляют собой пары вида

```
<название_параметра> = <значение>
```

Строки, начинающиеся с символа #, считаются комментариями и игнорируются.

Общие параметры для всех режимов работы серверной части:

prHost	Имя хоста, на котором работает процесс-координатор сервера задач
prPort	Порт, на котором сервер задач принимает соединения от сервиса запросов
prSock	Сокет для общения сервера задач и сервиса запросов. Поддерживается только для ОС UNIX/Linux. Используется как альтернатива общению через TCP/IP. В файле конфигурации должен присутствовать либо сокет (prSock), либо пара хост:порт (prHost и prPort).
ssHost	Имя хоста, на котором работает сервис запросов
ssPort	Порт, на котором сервис запросов принимает соединения от вычислительных клиентов
ssREQHost,	Хост и порт, на котором работает дополнительный сервис запросов REQ

ssREQPort	(запрос клиентами очередной порции). Необязательные параметры, значения задаются аналогично ssHost и ssPort.
ssASWHost, ssASWPort	Хост и порт, на котором работает дополнительный сервис запросов ASW (отправка результатов на сервер). Необязательные параметры, значения задаются аналогично ssHost и ssPort.
taskAPI	Определяет режим работы серверной части. Значения: - Perl – основной сервер, прикладная задача реализована с интерфейсом Perl API; - Files – основной сервер, прикладная задача реализована с интерфейсом Files API; - Proxy – промежуточный сервер; - Subserver – сервис запросов
dlSites	URL (если их несколько, то список пишется через пробел), по которому вычислительный клиент скачивает себе клиентскую часть задачи и дополнительные файлы (если таковые требуются). Необязательный параметр; по умолчанию принимает значение http://<ssHost>:<ssPort>, представляющее собой файловый сервис запущенного сервера. Могут указываться и "зеркала" – альтернативные файловые сервисы, в качестве которых могут использоваться любые http-серверы.
dlFiles	Список (через пробел) вспомогательных файлов, которые вычислительный клиент может потребовать и скачать на узлы. Необязательный параметр.
dlOrder	Определяет порядок, в котором клиент использует URL из списка в качестве файлового сервиса. Принимает значения: - Random – берется случайный URL из списка - Fallback – URL берутся последовательно При наличии только одного значения параметра dlSites параметр dlOrder может иметь любое значение. Необязательный параметр, по умолчанию принимается значение "Fallback".
logFile	Файл, в который будет записываться протокол (лог-файл) работы серверной части. Необязательный параметр. По умолчанию лог-файлы сохраняются в gserv/logs/<дата_и_время_начала_расчета>
keepFirst	Определяет логику поведения сервера задач при повторном получении результатов уже обработанной порции. Значения: -1 – повторно полученная порция не передается серверной части прикладной задачи, - 0 – серверной части задачи передаются абсолютно все порции. Необязательный параметр. Значения по умолчанию: 1 во всех режимах, кроме промежуточного сервера, и 0 для промежуточного сервера. Целесообразно выставлять значения, отличное от принятого по умолчанию в том случае, если, например, серверная часть задачи осуществляет контроль корректности вычислений, и получение одинаковых результатов для нее существенно.
singlePortion	Необязательный параметр, определяющий логику повторной выдачи порций. Значения: - 0 (по умолчанию) – после выдачи порции с последним номером начинается раздача порций, результаты которых еще не получены - 1 – каждая порция выдается только 1 раз.
portionWait	Необязательный параметр, имеющий смысл только в режиме однократной раздачи порций сервером (singlePortion = 1). Определяет время (в секундах) ожидания ответа клиентов. Сервер завершит работу по истечении указанного времени, прошедшего с момента последней выдачи порции. Значение 0 (по умолчанию) – сервер не будет завершать работу.

	В текущей версии завершение работы поддерживается только в ОС Linux!
ssTempFiles	Необязательный параметр, определяющий необходимость использования временных файлов при формировании порций. Использование временных файлов оправдано, если порции имеют большой размер. Значения: - 0 (по умолчанию) – не использовать временные файлы, - 1 – использовать. Временные файлы сохраняются в подкаталоге gserv/tmp
ssTempDir	Необязательный параметр, определяющий каталог хранения временных файлов. По умолчанию используется xcom/gserv/tmp. Данный каталог используется не только при явном указании ssTempFiles, но и неявно, например, при использовании API Files и сокетов для общения процессов серверной части.

В режиме работы серверной части как основного либо промежуточного сервера значения параметров prHost и ssHost совпадают и являются именами хост-машины, на которой запускается серверная часть X-Com[2]. Клиенты обращаются к серверной части по адресу http://<ssHost>:<ssPort>

Параметры для работы в режиме основного сервера (общие для API Files и Perl):

taskName	Имя задачи
taskId	Идентификатор (номер) задания. Применяется в основном при работе совместно с подсистемой управления заданиями. Для одиночных ручных запусков серверной части допустимы любые целочисленные значения. Необязательный параметр, по умолчанию выставляется значение 1.
taskArgs	Список аргументов, передаваемых прикладной задаче. Необязательный параметр.
taskDesc	Описание задачи (используется для визуализации хода расчета). Необязательный параметр.

Параметры для работы в режиме основного сервера (только для Files API):

taskList	Список входных файлов к обработке. В каждой строке – имя одного файла (каждый файл считается одной вычислительной порцией), после которого через пробел могут перечисляться дополнительные параметры, которые будут переданы клиенту в конце общих параметров задачи, указанных в taskArgs. При задании параметра taskIn файлы будут браться из указанного каталога, в противном случае должны быть прописаны полные пути к файлам.
taskIn	Путь к входным файлам для задачи. Необязательный параметр. При его наличии файлы, перечисленные в списке из taskList, будут искаться в указанном каталоге
taskOut	Каталог, в который будут сохраняться выходные файлы задачи
outExtension	Расширение, которое будут иметь выходные файлы. Необязательный параметр. Значение по умолчанию ".out".

Параметры для работы основного сервера при использовании интерфейса Processes в клиентской части задачи:

xCliAPI	Может принимать значения Perl (значение по умолчанию, работа с использованием файла gctask) и Processes (указание команд инициализации и обработки данных, а также имен файлов). Нижеследующие параметры приведены для интерфейса Processes.
xCliInitCmd	Командная строка, которая будет вызвана для инициализации клиентской части
xCliPortionCmd	Командная строка, которая будет вызываться для обработки каждой

	порции
xCliPortionIn	Имя файла, в который будет записана очередная входящая порция. Если параметр не указан, будет использоваться имя portion.in.
xCliPortionOut	Имя файла, в который будет сохранен результат обработки порции. Если параметр не указан, будет использоваться имя portion.out.
xUser	Пользователь, от имени которого будет выполняться запуск процессов на стороне клиента

Параметры для работы в режиме промежуточного сервера:

upHost	Имя хоста вышестоящего сервера
upPort	Номер порта, на котором вышестоящий сервер принимает запросы от вычислительных клиентов
httpProxy	Адрес http-proxy (при необходимости работы через него)
proxyTmp	Временный каталог, куда записываются буферизированные порции. Необязательный параметр. По умолчанию порции сохраняются в gserv/proxy.
inBufferMax	Максимальное число входящих порций, буферизуемых промежуточным сервером
inBufferMin	Минимальное число порций во входном буфере, по достижении которого промежуточный сервер пополнит буфер
outBufferMax	Максимальный размер буфера исходящих порций
outBufferMin	Число исходящих порций, отправляемых вышестоящему серверу в одном запросе
clId	"Идентификатор клиента", сообщаемый промежуточным сервером вышестоящему
clOS	Тип операционной системы нижележащих узлов (допустимые значения перечислены в п.4.3. настоящего руководства)
clCPUType	Тип CPU нижележащих узлов (допустимые значения перечислены в п.4.3. настоящего руководства)
clCPUCores	Число процессорных ядер нижележащих узлов
clCPUFreq	Частота CPU нижележащих узлов (в герцах)
clCPUPerf	Производительность одного ядра узла (в флопсах)
clRAMSize	Объем оперативной памяти нижележащих узлов (в байтах)

Параметры clOS, clCPUType, clCPUCores, clCPUFreq, clCPUPerf и clRAMSize служат для того, чтобы промежуточный сервер получил от вышестоящего сервера исполнимый код прикладной задачи, оптимизированной именно для таких параметров. Фактически промежуточный сервер выдает представляется вышестоящему серверу узлом, обладающим перечисленными свойствами.

Во время работы серверной части можно просматривать общее состояние хода вычислений с помощью веб-интерфейса, доступного по адресу:

```
http://<ssHost>:<ssPort>/status.html
```

(хост и порт те же, что и для вычислительных клиентов).

Полученная страница (Рис. 2) обновляется автоматически и содержит данные о количестве посчитанных и оставшихся порций, а также сведения об узлах, подключенных к серверу (имя хоста, платформа, номер текущей порции, время счета текущей порции, среднее время счета порции на узле). Легенда приведена в нижней части отображаемой страницы.

Аналогичные данные доступны и формате XML. Для их получения необходимо обратиться по URL

```
http://<ssHost>:<ssPort>/status.xml
```

Пример простейшего файла настройки сервера для работы с задачей Power (см. раздел "Тестовые задачи"):

```
# Server info

prHost      = localhost             # внутренний хост сервера задачи
prPort      = 65003                 # внутренний порт сервера задачи
ssHost      = localhost             # хост для клиентов
ssPort      = 65002                 # порт для клиентов

# Task info

taskName    = Power                 # имя задачи
taskId      = 10                    # номер задачи
taskAPI     = Perl                  # API задачи
taskArgs    = 2                     # аргументы задачи
```

В данном примере используется "имя хоста" внутреннего интерфейса localhost, т.е. все обращения к серверу (веб-браузеров или клиентов) будут работать только с того же компьютера, на котором запущен сервер. URL для клиентов X-Com[2] будет http://localhost:65002, URL для просмотра информации о ходе расчета в браузере – http://localhost:65002/status.html.

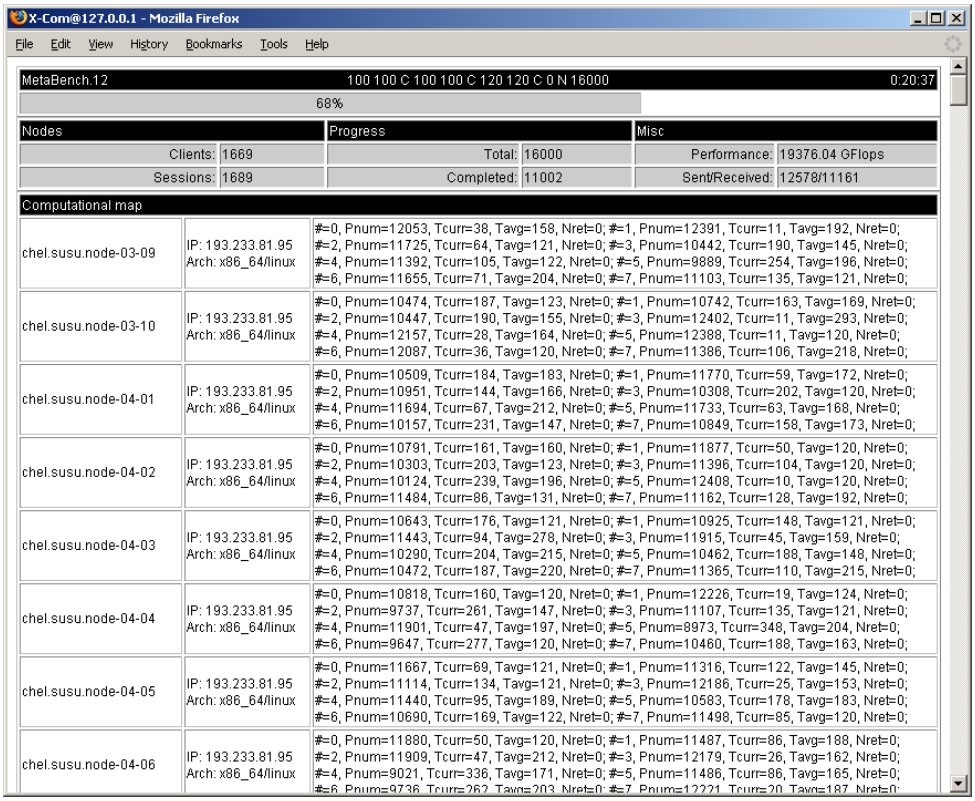

Рис.2. Пример отображения данных о ходе расчета в веб-браузере

Чтобы сервисы стали доступны в сети, необходимо указывать внешние адреса сетевых интерфейсов (доменные имена или IP-адреса). В этом случае секция "Server info" будет иметь примерно следующий вид:

```
# Server info

prHost      = xcom.host.ru          # внутренний хост сервера задачи
prPort      = 65003                 # внутренний порт сервера задачи
ssHost      = xcom.host.ru          # хост для клиентов
ssPort      = 65002                 # порт для клиентов
```

Здесь xcom.host.ru – имя машины, на которой запущен сервер. К серверу с такими настройками клиенты должны обращаться с URL http://xcom.host.ru:65002, веб-браузеры – с URL http://xcom.host.ru:65002/status.html.

Чуть более сложный пример – настройка сервера для задачи TestPl:

```
# Server info

prHost            = xcom.host.ru
prPort            = 65003
ssHost            = xcom.host.ru
ssPort            = 65002

# File download service

dlSites           = http://xcom.host.ru:65002/ http://mirror.host.ru/
dlOrder           = Random

# Task info

taskName          = TestPl
taskId            = 1
taskAPI           = Perl
taskArgs          = 100 100 C 100 100 C 0 1 R 0 N 10
taskDesc          = Dummy task for Perl API testing
```

В данном примере появляется секция "File download service", в которой указываются, что файлы с клиентской частью прикладной задачи могут браться как с встроенного файлового сервиса http://xcom.host.ru:65002/, так и с дополнительного "зеркала" http://mirror.host.ru/ (которым может служить любой http-сервер), причем конкретный адрес будет выбираться случайно.

Пример настройки сервера для задачи, использующей интерфейс Files API и дополнительную загрузку файлов на узлы (секции "Server info" и "File download service" могут быть взять из приведенных выше примеров):

```
# Task info

taskName          = files-test
taskId            = 2
taskAPI           = Files
taskArgs          = -a 100
taskList          = /home/user/xcom/tasks/files-test/in/list
taskIn            = /home/user/xcom/tasks/files-test/in
taskOut           = /home/user/xcom/tasks/files-test/out
outExtension      = .tar.gz
```

```
# Data info

dlFiles           = /home/user/xcom/tasks/files-test/library.bin
```

В данном примере список входных файлов задан в файле /home/user/xcom/tasks/files-test/in/list, сами входные файлы находятся в каталоге /home/user/xcom/tasks/files-test/in, результаты обработки будут сохранены в каталоге /home/user/xcom/tasks/files-test/out в файлах, имена которых состоят из имен исходных файлов с приписанными к ним "расширениями" .tar.gz. Клиент X-Com^2 также автоматически загрузит на узел файл library.bin, который может быть использован прикладной задачей.

Серверная часть X-Com поддерживает распределение своих процессов по физически различным компьютерам. На одной из машин запускается процесс-координатор – фактически это обычный запуск серверной части X-Com с указанием, на каких хостах будут располагаться сервисы запросов, общающиеся с клиентами. На других машинах серверная часть X-Com запускается в режиме Subserver, т.е. без процесса-координатора.

Рассмотри пример запуска серверной части X-Com на 3-х машинах. На узле с IP-адресом 10.1.36.2 запустим процесс-координатор с сервисами запросов, которые будут обслуживать только запросы типа TSR и загрузку файлов:

```
prHost      = 10.1.36.2       # здесь работает процесс-координатор
prPort      = 65008           # (сервер задачи)

ssHost      = 10.1.36.2       # с этого адреса клиент запросит описание
ssPort      = 65009           # задачи и файлы

ssREQHost   = 10.1.36.3       # с этого адреса клиенты будут запрашивать
ssREQPort   = 65009           # очередную порцию

ssASWHost   = 10.1.36.4       # на этот адрес клиенты будут отправлять
ssASWPort   = 65009           # результаты
```

На узлах 10.1.36.3 и 10.1.36.4 запустим серверную часть X-Com в режиме Subserver:

```
prHost      = 10.1.36.2       # общаемся с процессом-координатором
prPort      = 65008           # по указанному адресу

ssHost      = 10.1.36.4       # по этому адресу обслуживаем клиентов
ssPort      = 65009           #

taskAPI     = Subserver       # работаем в режиме сервиса запросов
```

Заметим, что процессы сервисов запросов, запущенные на любой машине, фактически равноправны. Любой сервис запросов корректно обработает и передаст процессу-координатору любой пришедший к нему клиентский запрос. В приведенном выше примеры клиенты могли бы полноценно работать только с одной из машин, однако указание параметров ssREQHost и аналогичных заставляет клиентов использовать конкретные адреса для отправки заданных запросов.

6. Запуск вычислительного клиента X-Com^2

Модули вычислительного клиента расположены в каталоге xcom/gcli. Для запуска клиента используется файл client.pl. Если этот файл не имеет флага выполнимости (x-bit) и для запуска используется интерпретатор perl, то запуск должен осуществляться командной строкой вида

```
perl -T <другие флаги интерпретатора> <путь к файлу client.pl>
```

Если файл client.pl находится в текущем каталоге, то <путь к файлу client.pl> должен иметь вид

```
./client.pl
```

Клиент поддерживает следующие параметры командной строки:

-s <URL>	URL серверной части X-Com^2 (сервиса запросов или менеджера клиентов), по которому можно получить описание задачи. Обязательный параметр
-c <имя>	Задает идентификатор клиента. Если этот параметр не задан, используется имя узла, получаемой при помощи метода hostname стандартного модуля Perl Sys::Hostname
-d	Включает режим отладки. При работе в режиме отладки при аварийном завершении клиента на экран выдается стек вызовов подпрограмм
-T <каталог	Задает рабочий каталог для размещения пользовательских файлов. Если этот параметр не задан, временный подкаталог с именем вида x-com-client-XXXX создается в системном каталоге для временных файлов
--http-proxy <URL>	Адрес http-proxy
--add-arg <arg1…argN>	Дополнительные аргументы, передающиеся в клиентскую часть задачи (в функции gcprepare и gctask)

Клиент X-Com^2 работает в режиме taint. В этом режиме Perl помечает все данные, полученные из внешних источников (ввод программы, файлы, сеть и т.п.) как опасные (tainted) и не разрешает передавать их наружу программы. Результаты выражений с участием опасных данных тоже являются опасными. Чтобы снять с данных пометку опасности, надо проверить их соответствие регулярному выражению и взять часть этого выражения ($1). Подробнее см. на странице perlsec (http://perldoc.perl.org/5.8.8/perlsec.html).

Клиентская часть прикладной задачи, исполняемая внутри того же экземпляра интерпретатора Perl, что и код клиента, также должна быть предназначена для работы в режиме taint. Также при запуске клиентской части прикладной задачи удаляются следующие переменные окружения: IFS, CDPATH, ENV, BASH_ENV и PATH. Это также следует учитывать при подготовке клиентской части прикладной задачи.

Рекомендуется именовать клиентов (задавать идентификаторы) 4-звенными именами, например, по следующему принципу:

```
<город>.<имя_кластера>.<имя_узла>.<номер_клиентского_процесса_на_узле>
```

Например:

```
msk.skif_mgu.node-01-01.1
```

При использовании внешних систем визуализации хода расчета такой принцип позволяет группировать узлы по городам, кластерам и т.д. Также группировка по кластерам используется скриптом анализа лог-файлов сервера для получения сводной статистики (см. далее).

7. Использование подсистемы управления заданиями

Подсистема управления заданиями XQSERV предоставляет удобный (но не обязательный) способ работы с задачами в распределенной среде на основе системе метакомпьютинга X-Com^2. XQSERV принимает задания от пользователей, формирует из них очередь, запускает последовательно одно или более заданий из очереди, отслеживает их состояние и перенаправляет клиентов X-Com^2 к соответствующим серверам задач. Сервер XQSERV должен быть запущен постоянно, клиенты XQSERV вызываются пользователями в командной строке. Для постановки задания в очередь пользователи формируют файл описания задания в формате, аналогичном формату конфигурационных файлов сервера X-Com^2. Таким образом, пользователи избавляются от забот по самостоятельной о настройке и запуске серверов и могут пользоваться доступными распределенными ресурсами в рамках организованного сервиса.

XQSERV позволяет указать максимальное число одновременно запущенных заданий. Если данное число будет равно 1, задания будут запускаться последовательно. Если же данный параметр будет иметь значение 2 и более, одновременно будет запущено соответствующее число серверов X-Com^2. При запуске клиентов X-Com^2 в качестве адреса сервера необходимо указывать адрес машины, на которой запущен сервис XQSERV с номером порта менеджера клиентов (см. далее).

Текущая версия XQSERV поддерживает два основных метода распределения клиентов к запущенным заданиям. Первый метод – равномерное распределение. XQSERV старается распределять клиентов поровну к каждой запущенной задаче. Второй метод – распределение с учетом требований прикладной задачи к ресурсам клиента (тип, частота и производительность процессора, объем оперативной памяти и т.д.). В этом случае клиенты направляются на решение только тех задач, требованиям которых они удовлетворяют. Если имеется несколько одновременно работающих задач, и подключающийся клиент удовлетворяет требованиям каждой из них, он будет перенаправлен к задаче, запущенной раньше всех.

Если в среде запускается не более одной задачи в каждый момент времени, а механизм требований прикладной задачи не используется, то работать через менеджер клиентов нецелесообразно, поскольку он всегда будет перенаправлять клиентов к серверу задач с одним и тем же адресом. В этом случае для клиентов X-Com^2 можно явно указывать хост и порт сервера X-Com^2, задаваемые в настройках XQSERV.

Сервер и клиенты XQSERV общаются между собой либо через Интернет поверх протокола HTTP, либо через сокеты UNIX (не поддерживается в ОС MS Windows). Первый способ более универсальный (сервер и клиенты могут находиться на разных физическим машинах, работающих под управлением различных ОС). Второй способ обеспечивает более высокий уровень безопасности, поскольку имя пользователя, запускающего клиента системы очередей, определяется системными средствами. Однако в этом случае и клиент, и сервер XQSERV должны работать на одной физической машине.

Модули подсистемы управления заданиями XQSERV расположены в каталоге xcom/xqserv.

7.1. Запуск управляющего сервера и менеджера клиентов

Для настройки управляющего сервера и менеджера клиентов (сервера подсистемы управления заданиями) используются файл конфигурации, аналогичный файлам настройки

серверной части X-Com^2. Запуск сервера подсистемы управления заданиями осуществляется командой

```
./XQServ.pl [config.ini]
```

в Linux, или

```
XQServ.cmd [config.ini]
```

в MS Windows (напоминаем, что для работы требуется модуль Win32::Process). При запуске без параметров считывается конфигурационный файл XQServ.ini, который должен находиться в том же каталоге, что и исполнимый файл XQSERV. В качестве параметра командной строки можно указать путь к (альтернативному) файлу конфигурации.

Строки файла конфигурации представляют собой пары вида:

```
<название_параметра> = <значение>
```

Строки, начинающиеся с символа #, считаются комментариями и игнорируются.

Параметры конфигурационного файла сервера подсистемы управления заданиями:

XQServHost	Имя хоста, на котором сервер подсистемы управления заданиями принимает HTTP-запросы от клиентов или веб-браузеров
XQServPort	Порт, на котором сервер принимает HTTP-запросы
XQServSocket	Имя сокета UNIX для общения сервера XQSERV с клиентами. Необязательный параметр. Если он задан, то клиенты должны работать с сервером только через указанный сокет, а по HTTP можно получить лишь статистику с помощью веб-браузера. Если сокет не задан, то вся работа ведется только через HTTP.
XQServSocketPerm	Права, с которыми создается описанный выше сокет. Задается в виде 8-ричного значения вида 0xyz. По умолчанию задано значение 0770.
CliManagerHost	Имя хоста, на котором работает менеджер клиентов (может совпадать с XQServHost)
CliManagerPort	Порт, на котором менеджер принимает соединения от вычислительных клиентов
XcomServerPath	Каталог, в котором расположена серверная часть X-Com^2
XcomServerName	Имя скрипта запуска серверной части X-Com^2 (может иметь значение, отличное от "Xserv.pl", только в случае, если используются нестандартные варианты X-Com^2)
XcomMinPort	Наименьший номер порта, начиная с которого порты будут использоваться при запуске серверов X-Com^2.
XcomPrHost XcomSsHost	Эти параметры используются для запуска сервера задач и полностью аналогичны параметрам файлов конфигурации серверной части X-Com^2 prHost и ssHost. Параметр prHost указывается в том случае, если общение между компонентами сервера X-Com^2 будет вестись поверх протокола TCP. Если необходимо организовать общение процессов серверной части X-Com^2 посредством UNIX-сокетов, используется параметр XcomPrSocketPath (см. ниже).
XcomPrSocketPath	Каталог, в котором будут размещены сокеты для серверной части X-Com^2 (например, /tmp). В файле конфигурации должен присутствовать один из двух параметров – XcomPrHost или XcomPrSocketPath.
XcomSinglePortion	Серверы X-Com^2 будут запущены с параметром singlePortion и заданным значением
XcomPortionWait	Серверы X-Com^2 будут запущены с параметром portionWait и заданным значением. Имеет смысл только при наличии параметра XcomSinglePortion.
XcomWriteUser	Определяет, передавать ли имя пользователя, запустившего клиент xqc, серверу задач. По умолчанию имя пользователя не передается, при

	установке значения "1" имя пользователя будет передано в параметре xUser файла настройки серверной части X-Com.
MaxRunningTasks	Максимальное число одновременно работающих заданий
ClientsRedirect	Параметр, определяющий логику распределения клиентов к запущенным серверам задач. Может принимать значения: - Equal (по умолчанию) – клиенты распределяются поровну к каждой работающей задаче; - Conditions – клиенты распределяются с учетом требований задач. Описание возможных требований см. в разделе "Файл описания задания".
KillXComOnExit	Определяет, останавливать или нет работающие серверы задачи X-Com, запущенные подсистемой управления заданиями, после прекращения работы сервера XQServ. По умолчанию процессы X-Com не останавливаются. Установка данного параметра в значение "1" обеспечивает остановку всех запущенных серверов X-Com.

Если MaxRunningTasks = 1, т.е. одновременно будет работать только одно задание, то клиентов X-Com можно настраивать на адрес http://<XComSsHost>:<XComMinPort>. В противном случае должен быть использован сервис переадресации клиентов, и клиенты настраиваются на адрес http://<CliManagerHost>:<CliManagerPort>.

По URL http://<XQServHost>:<XQServPort> доступен веб-интерфейс для просмотра состояния заданий из очереди в веб-браузере.

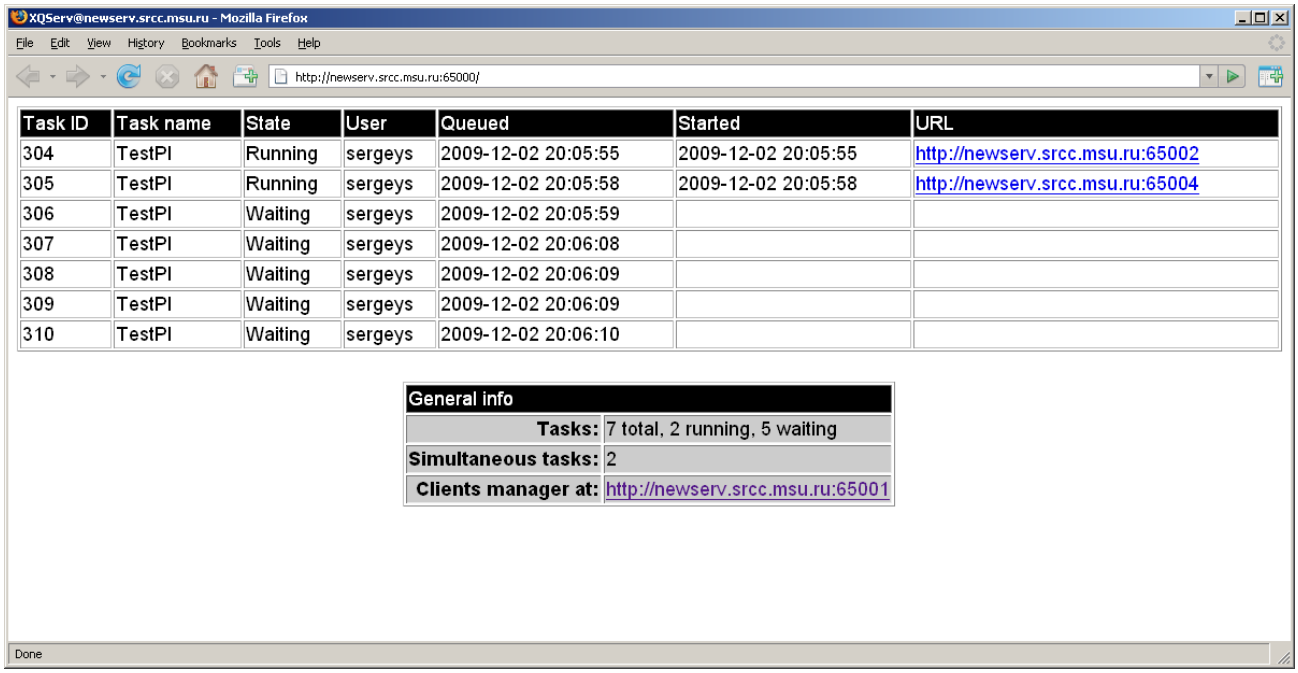

Рис. 3. Пример отображения очереди заданий подсистемы XQSERV

Пример файла xqserv.ini:

```
# XQServ server settings

XQServHost        = xcom.host.ru
XQServPort        = 65000
CliManagerHost    = xcom.host.ru
CliManagerPort    = 65001
```

```
XComServerPath    = D:\xcom\gserv
XComServerName    = XServ.pl

# X-Com server defaults

XComMinPort       = 65002
XComPrHost        = xcom.host.ru
XComSsHost        = xcom.host.ru
MaxRunningTasks   = 1
```

В данном примере пользовательские клиенты должны обращать к серверу XQSERV по адресу xcom.host.ru:65000, а клиенты X-Com^2 должны использовать URL http:// xcom.host.ru:65001 (при этом, поскольку выполняться одновременно может только один сервер X-Com^2, то номера портов сервера меняться не будут, и допустимо настраивать клиентов на URL http://xcom.host.ru:65002).

7.2. Запуск пользовательского клиента

Запуск пользовательского клиента осуществляется командой

```
./xqc.pl <команда>
```

в Linux, или

```
xqc.cmd <команда>
```

в MS Windows, где <команда> может принимать одно из следующих значений:

add <task.x>	Добавление в очередь задания, описанного в файле task.x После имени файла допускается указание пар вида <параметр> <значение> которые будут интерпретированы аналогично соответствующим параметрам из файла описания задания и будут иметь приоритет над параметрами из этого файла.
kill <ID>	Удаление из очереди или останов задания с номером ID
show	Просмотр состояния очереди

При запуске пользовательский клиент считывает конфигурационный файл xqc.ini (из того же каталога, в котором находится сам) со следующими обязательными параметрами:

socket	Сокет UNIX, который используется для общения с сервером XQSERV

или

host	Имя хоста, на котором принимает HTTP-запросы сервер XQSERV
port	Порт, на котором принимает HTTP-запросы сервер XQSERV

Клиент поддерживает общение либо через сокеты UNIX, либо через HTTP, поэтому в конфигурационном файле может быть задан либо единственный параметр socket, либо пара host и port.

Дополнительные (необязательные) параметры файла xqc.ini:

copy	Каталог, в который клиент xqc скопирует файл, указанный в параметре xFile файла описания задания (см. далее)

Пример файла xqc.ini:

```
host  = xcom.host.ru
port  = 65000
```

7.3. Файл описания задания

Файл описания задания строится по тем же принципам, что и конфигурационные файлы серверной части системы X-Com^2, и может содержать все параметры, относящиеся к прикладной задаче, реализованной с API Perl или Files: taskName, taskAPI, taskArgs и т.д. Параметр taskId – идентификатор задания в очереди – будет присвоен автоматически.

Пример простейшего файла описания задания:

```
taskName      = TestPl
taskAPI       = Perl
taskArgs      = 100 200 R 100 200 R 5 10 R 0 N 10
```

В файле описания задания также могут присутствовать требования прикладной задачи к вычислительным ресурсам, на которых она будет запускаться (если сервер XQSErv запущен в соответствующем режиме). Если задано одновременно несколько требований, то задача отправится на узел, удовлетворяющий всем требованиям (объединение по условию "И").

Поддерживаются следующие требования:

nodeMinCPUFreq nodeMaxCPUFreq	Минимальная и максимальная тактовая частота процессора (в герцах)
nodeMinCPUPerf nodeMaxCPUPerf	Минимальная и максимальная производительность процессора (в флопсах)
nodeMinCPUCores nodeMaxCPUCores	Минимальное и максимальное число процессорных ядер на узле
nodeMinRAMSize nodeMaxRAMSize	Минимальный и максимальный объем оперативной памяти узла (в байтах)
nodeCluster	Список (через пробел) префиксов имен узлов, на которые может быть распределена задача
nodeOS	Список (через пробел) допустимых операционных систем узла
nodeCPUType	Список (через пробел) допустимых процессорных архитектур узла

Дополнительные параметры, которые могут присутствовать в файле описания задачи

xFile	Файл, который клиент xqc скопирует в каталог, заданный параметром copy файла xqc.ini. К имени файла будет приписан уникальный набор символов (таким образом, одноименные файлы не будут затерты). При запуске сервера задач X-Com данное имя файла будет вставлено первым параметром в taskArgs.

Примеры:

```
nodeMinCPUFreq    = 1000000000      # допустимая частота CPU от 1 ГГц…
nodeMaxCPUFreq    = 4000000000      # … до 4 ГГЦ

nodeMinCPUPerf    = 4000000000      # производительность CPU от 4 GFlops…
nodeMaxCPUPerf    = 8000000000      # … до 8 GFlops

nodeMinRAMSize    = 4000000000      # объем памяти узла от 4 Гб…
nodeMaxRAMSize    = 8000000000      # … до 8 Гб
nodeMinCPUCores   = 4   # процессорных ядре на узле от 4…
nodeMaxCPUCores   = 8   # … до 8

nodeCluster       = msu.srcc msk.srcc     # список допустимых кластеров
nodeOS            = MSWin32 linux         # список допустимых ОС
```

```
nodeCPUType       = x86 x86_64      # список допустимых архитектур узлов
```

Если сервер XQServ запущен в режиме учета требований задач, при этом в файле описания задания никаких требований не задано, то считается, что такой задаче удовлетворяет любой клиент.

8. Тестовые задачи

8.1. Задача Pi

Задача предназначена для вычисления числа Пи методом Монте-Карло. Алгоритм вычисления имеет следующее математическое обоснование.

Пусть $Xn = (x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)$ – n пар независимых равномерно распределенных случайных величин. Пусть количество пар, для которых $(x_i^2+y_i^2) < 1$, равно K_{Xn}. Тогда последовательность $4K_{Xn}/n$ при $n\to\infty$ сходится по вероятности к числу Пи. При достаточно больших n вычисленное значение будет иметь точность до i-го знака после запятой, где $i = [-\log_{10}(2/(3n)^{1/2})]-1$.

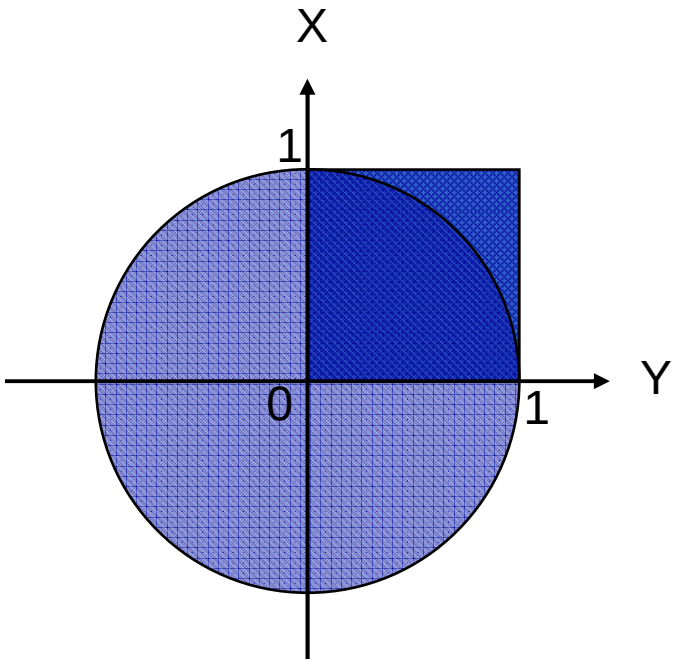

Рис. 4. Геометрическая интерпретация алгоритма вычисления числа Пи

Алгоритм также имеет простую геометрическую интерпретацию. Рассмотрим круг радиуса 1 и квадрат со стороной, равной 1, одна из вершин которого лежит в центре круга. Внутрь квадрата будем случайным образом бросать точки, часть из которых, очевидно, окажется внутри сектора круга. При достаточно большом числе бросков отношение вероятностей попадания точек в сектор или в квадрат будет равно отношению площадей фигур. В данном случае площадь квадрата равна 1, площадь сектора равна π/4. Отсюда число Пи вычисляется как умноженное на 4 отношение общего числа бросков к числу попаданий в сектор.

Задача Pi имеет два обязательных параметра, задающих мощность выборки пар случайных чисел для метода Монте-Карло:

1. N - количество порций,
2. E - задает мощность выборки 10^E для одной порции.

Таким образом, общая мощность выборки будет составлять 10^EN, при этом отклонение вычисленного значения при достаточно больших E и N составит $2/(10^{E/2}(3N)^{1/2})$.

Клиентская часть задачи в качестве порции данных получает от серверной части X-Com значение затравки для генератора случайных чисел (это номер порции), генерирует выборку заданной мощности и подсчитывает число попаданий внутрь круга, возвращая полученное число, умноженное на 4, в качестве результата обработки порции. Серверная часть X-Com суммирует результаты обработки порций и вычисляет число Пи по формуле $S/(10^EN)$, где S — сумма полученных результатов, N и E — параметры задачи Pi. В состав клиентской части входит исполнимый модуль, написанный на Си (исходный код прилагается), который вызывается из gctask с параметрами $portion $expo, где $portion — номер текущей порции, $expo — значение параметра задачи E.

Файл TestPi.pm содержит в себе реализацию серверной части TestPi. Функция addPortion при получении результатов обработки последней порции вычисляет оценку числа Пи и записывает полученное число в файл out/pi_<N>_<E>.txt. Также при получении результатов очередной порции в файл out/pi_esteem_<N>_<E>.csv в формате CSV (comma-separated values) записываются следующие данные:

- Portions – номер текущей порции,
- Drops – число бросков на момент получения порции,
- EstimatedPi – очередное приближение числа Пи,
- RealPi – реальное число Пи (приведено для сравнения),
- Error – ошибка, т.е. отклонение вычисленного числа Пи от реального.

Данный файл можно открыть с помощью MS Excel или другой электронной таблицы и использовать для построения графиков (см. пример на Рис. 4).

Пример фрагмента файла настройки сервера для задачи Pi:

```
#Task info
taskName    = Pi
taskId      = 1
taskAPI     = Perl
# task Args = N E
# the number of random drops will be N*(10**E)
taskArgs    = 100 8
# task description
taskDesc    = Computing Pi using Monte-Carlo method
```

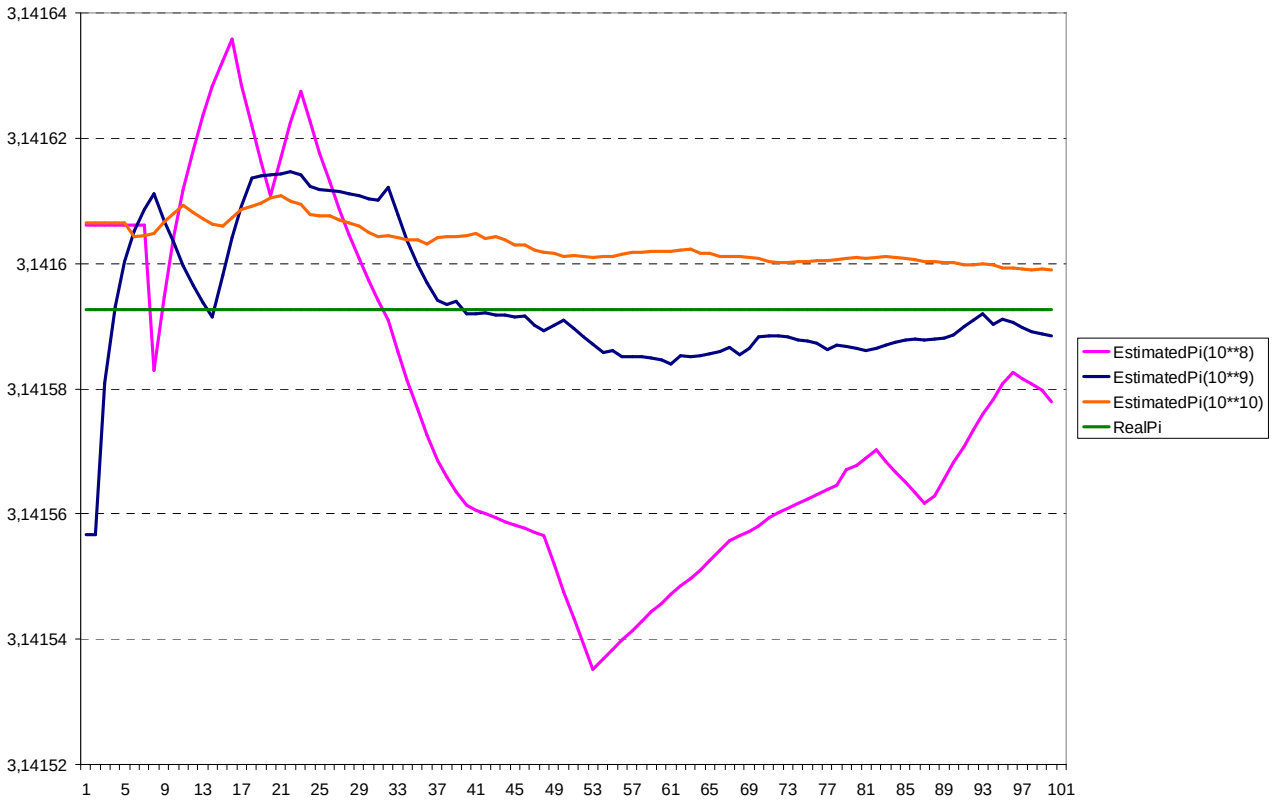

Рис. 5. Пример графика приближения к числу Пи при N = 100 и различных значениях E

Параметры задачи Pi можно подобрать наилучшим образом в зависимости от свойств вычислительной среды. Если известно количество клиентов, и они однородны, то наиболее сбалансированной нагрузки для выполнения Pi можно достичь, указав N кратным количеству клиентов и задав параметр E в соответствии с желаемой общей мощностью выборки. При таком задании параметров клиенты начнут обработку порций в одновременно (с точностью

до времени получения порции) и закончат её в одно время, что обеспечит постоянную загрузку всех вычислительных узлов во всё время работы задачи а также минимальное число коммуникаций, но при большом числе узлов характеристики коммуникационной среды и программно-аппаратного обеспечения сервера X-Com^2 могут негативно повлиять на время, затраченное на раздачу порций и прием результатов их обработки.

В случае неоднородной вычислительной среды параметру *E* следует задать малое значение, а *N*, соответственно, большое. Организовав таким образом Fine-Grained вычисления, можно достичь сбалансированной загрузки узлов, но затраты на коммуникации при этом возрастут в соответствии с числом порций

8.2. Задача Power

Задача Power включена в дистрибутив в качестве "заглушки" для разработки собственных программ на ее основе. Номинальная функция задачи Power – возведение в заданную степень (задается в качестве аргумента задачи) последовательности четных чисел от 1 до 100. Клиентская и серверная часть задачи подробно разобраны в соответствующих разделах настоящего руководства.

8.3. Задача TestPl

Задача TestPl предназначена для проверки работоспособности среды метакомпьютинга X-Com^2, для имитации реальных вычислений, оценки нагрузки на различные компоненты среды и т.д. Серверная часть задачи генерирует "входные данные" заданного размера, клиентская часть принимает эти данные, производит вычисления и отправляет "выходные данные" серверу. Сервер записывает полученные от клиентов данные в директорию xcom/tasks/TestPl/in - по одному файлу на каждую порцию. В дистрибутив включены версии задачи для Linux и MS Windows.

8.3.1. Параметры тестовой задачи

Тестовая задача имеет 12 обязательных параметров, отражающих различные варианты поведения модельной задачи (напоминаем, что входные параметры задачи прописываются в строке taskArg файла конфигурации сервера или файла описания задания):

1) мин. размер исходящих порций ("входные данные"), байт
2) макс. размер исходящих порций, байт
3) изменение размера исх. порций (возможные значения - символы C, R, I или D)
4) мин. размер входящих порций ("результаты"), байт
5) макс. размер входящих порций, байт
6) изменение размера вх. порций (возможные значения - символы C, R, I или D)
7) мин. время счета на узле, сек.
8) макс. время счета на узле, сек.
9) изменение времени счета (возможные значения - символы C, R, I, D или E)
10) размер матрицы для счетного процессорозагружающего модуля задачи
11) выбор счетного модуля (возможные значения - символы N, M, L или U)
12) число порций

Пояснение к параметрам, задающим алгоритм изменения размеров (пп.3,6):

- C (Constant) – размер не меняется и равен минимальному

- R (Random) – размер изменяется случайным образом в диапазоне между мин. и макс.
- I (Increase) – размер равномерно увеличивается от мин. до макс. значения
- D (Decrease) – размер равномерно уменьшается от макс. до мин. значения

Размеры задаются в байтах.

Для изменения времени счета порции (п.9) все аналогично. Добавляется параметр E - в этом случае задание не прерывается по истечении определенного времени, а выполняется до конца. Соответственно, время счета будет зависеть от выбранного счетного модуля и размера матрицы для него. Время задается в секундах.

8.3.2. Счетные модули

Задача содержит 3 различных счетных модуля, обеспечивающих загрузку процессора и резервирование памяти, а также поддерживает режим работы без загрузки процессора. Счетный модуль указывается 11-м параметром задачи и может быть одним из следующих:

- M (Matrix) - умножение 2-х квадратных матриц заданного размера. Элементы матрицы (по умолчанию типа int) генерируются случайным образом. Резервируется память для 3-х матриц.
- L (matrix_Light) - умножение матрицы заданного размера самой на себя. Резервируется память для 1-й матрицы.
- U (matrix_Ultralight) - имитация умножения матриц путем выполнения аналогичного числа арифметических операций над случайными числами. Память под матрицу не резервируется.
- N (None) - никакого счета не производится, выполняется sleep.

Если для изменения времени счета указывается значение, отличное от E, то счетный модуль по истечении определенного времени прерывает работу. В противном случае времена счета игнорируются, и модуль отрабатывает до своего естественного завершения.

8.3.3. Пример

```
# номера параметров:
#         1  2  3 4  5  6 7 8 9 10   11 12
taskArg = 20 20 C 40 40 C 1 5 R 1000 M 100
```

- входные данные по 20 байт на порцию, выходные данные по 40 байт на порцию, время счет изменяется случайным образом от 1 до 5 секунд, вычислительный модуль Matrix, размер матриц 1000x1000, 100 вычислительных порций.

9. Дополнительные утилиты

9.1. Анализатор лог-файлов сервера X-Com^2

Во время работы сервер X-Com^2 создает лог-файл, в котором фиксируются все запросы к серверу. По умолчанию лог-файлы сохраняются в каталоге xcom/gserv/log с именем <дата_и_время_запуска_сервера>, переопределить имя и путь можно с помощью параметра logFile в ini-файле сервера. Строки лог-файла – набор записей, разделенных символом табуляции:

- время записи события в лог-файл
- IP-адрес машины, с которой пришел запрос к серверу
  - в первой строке (START) вместо IP-адреса указывается имя файла настройки сервера
- тип запроса:
  - START – запуск сервера
  - STAT – запрос информации о ходе расчета
  - TFILE – запрос на скачивание клиентской части задачи
  - DFILE – запрос на скачивание вспомогательных файлов
  - TSR – запрос начальных данных клиентом
  - REQ – запрос клиентом очередной порции
  - ASW – ответ от клиента с результатами обработки порции
- параметры запроса. Различаются для каждого типа. Основные параметры:
  - STime – время начала обработки запроса
  - ETime – время окончания обработки запроса
  - File – имя скачиваемого файла (только для TFILE и DFILE)
  - Length – размер файла или порции
  - SId – идентификатор сессии клиента
  - Node – имя узла клиента
  - Portion – номера порций
  - Time – время обработки порции (-ий) на узле (только для ASW)
  - OS, CPUType, CPUNumCores, CPUFreq, CPUPerf, RAMSize – характеристики узлов (только для TSR)

Для автоматизированной обработки лог-файлов и получения статистической информации о завершенном расчете в состав дистрибутива системы X-Com^2 включена утилита xcom/utils/XLogProc.pl. Утилита принимает на вход имя лог-файла и создает в текущем каталоге набор файлов, содержащих информацию о ходе расчета, предназначенную как для чтения пользователем, так и для построения графиков с использованием стороннего ПО (MS Excel, OpenOffice Calc). Утилита позволяет управлять форматом выдачи данных – имеется возможность указания файла-шаблона, в котором будут перечислены нужные параметры (подробнее см. далее).

Запуск утилиты:

```
./XLogProc.pl <лог_файл> [опции]
```

Допустимые опции:

-i <секунды>	Интервал (в секундах), соответствующий шагу по времени в графиках, отображающих интенсивность различных запросов к серверу. По умолчанию - 60 секунд.
-t <файл_шаблона>	Шаблон для выдачи общей информации о расчете. По умолчанию читается шаблон из файла default_html.tpl, на основе которого строится

	файл в формате HTML.
-m <new|add>	Режим записи в создаваемые файлы: - add – файлы не перезаписываются, данные добавляются в конец существующих файлов (удобно для пакетной обработки нескольких лог-файлов, когда по заданному шаблону строится сводный отчет по серии расчетов) - new – файлы создаются или перезаписываются (по умолчанию).
-o <имя_файла>	Имя файла, в который будет записана общая информация. По умолчанию имеет вид <имя_задачи>.html
-ot <имя_файла>	Имя файла с табличными данными об интенсивности запросов, разбитых по временным интервалам. По умолчанию имеет вид <имя_задачи>.general.txt
-op <имя_файла>	Имя файла с табличными данными по каждой порции. По умолчанию имеет вид <имя_задачи>.portions.txt
-on <имя_файла>	Имя файла с табличными данными по каждому вычислительному узлу. По умолчанию имеет вид <имя_задачи>.nodes.txt

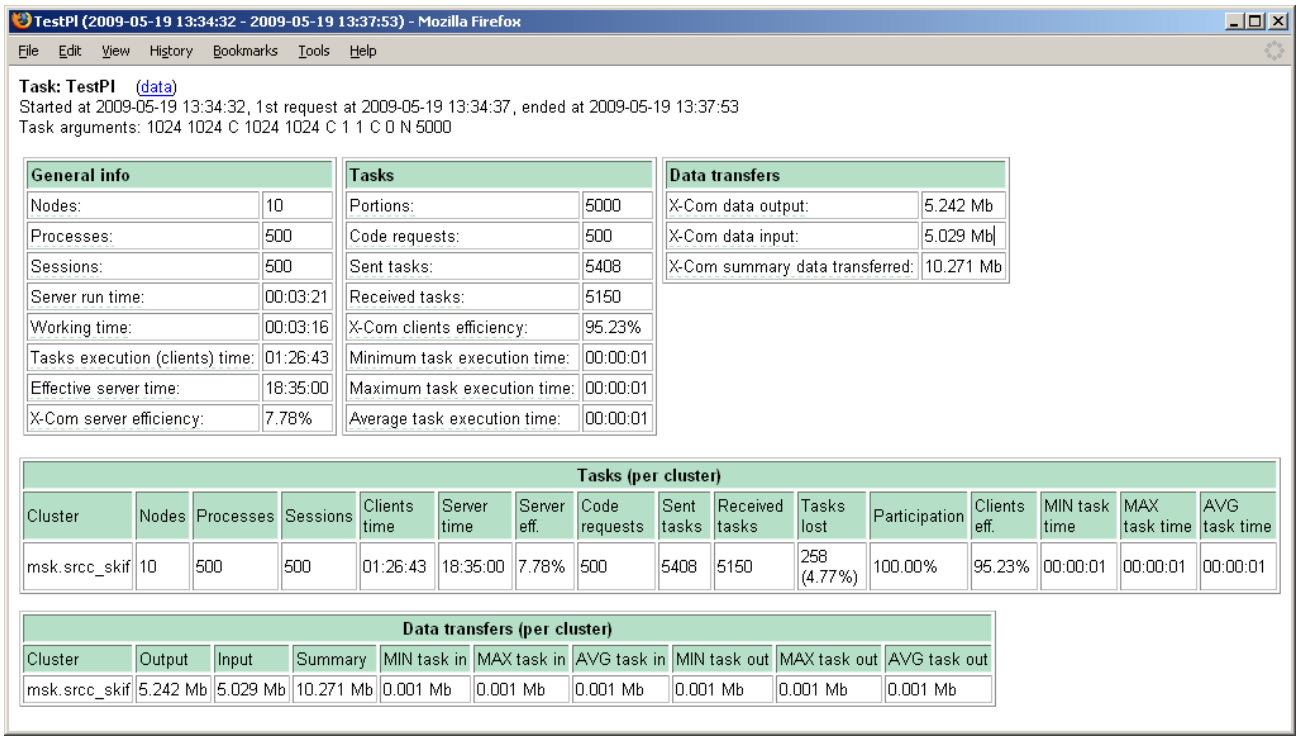

Рис.6. Пример отображения файла с общими данными о расчете

При запуске с параметрами по умолчанию создаются следующие файлы:

- <имя_задачи>.html – содержит общие сведения о расчете (данные о подключенных узлах, временны*е* данные, оценки эффективности расчета и т.д.). На Рис. 4 приведен пример отображения файла с общими сведениями.
- <имя_задачи>.general.txt – файл для анализа и/или построения графиков интенсивности запросов различных типов, приходящих к серверу. Имеет следующую структуру (столбцы данных разделяются символами табуляции):
  - число и время
  - кол-во запросов типа TSR (сведения о задаче) клиентами за указанный интервал времени, по каждому кластеру (столбцы "TSR,<имя_кластера>") и суммарно (столбец "TSR,SUM")
  - кол-во запросов типа REQ (получение клиентами порций), по каждому кластеру (столбцы "REQ,<имя_кластера>") и суммарно (столбец "REQ,SUM")

    - кол-во запросов типа ASW (ответы клиентов), по каждому кластеру (столбцы "ASW,<имя_кластера>") и суммарно (столбец "ASW,SUM")
    - активность клиентов (общее число запросов всех типов к серверу), по каждому кластеру (столбцы "ALL,<имя_кластера>") и суммарно (столбец "ALL,SUM")
    - объем данных (в байтах), отправленных сервером клиентам, по каждому кластеру (столбцы "OUT,<имя_кластера>") и суммарно (столбец "OUT,SUM")
    - объем данных (в байтах), отправленных клиентами серверу, по каждому кластеру (столбцы "IN,<имя_кластера>") и суммарно (столбец "IN,SUM")
    - суммарный объем трафика между (в байтах) между клиентами и сервером, по каждому кластеру (столбцы "INOUT,<имя_кластера>") и суммарно (столбец "INOUT,SUM")
- <имя_задачи>.portions.txt – файл для анализа и/или построения графиков по каждой порции. Структура:
    - Portion - номер порции
    - Node - идентификатор клиента, последним обработавшего такую порцию
    - ClientTime – время обработки порции на узле
    - ServerTime – время, прошедшее от начала выдачи до завершения получения и обработки этой порции сервером
    - REQLength - размер исходной порции
    - REQOverhead – накладные расходы сервера на формирование порции (разница между временами начала и завершения обработки)
    - ASWLength – длина результата обработки порции
    - ASWOverhead - накладные расходы сервера на обработку результирующей порции (разница между временами начала и завершения обработки)
- <имя_задачи>.nodes.txt – файл для анализа и/или построения графиков по каждому клиенту (считаются уникальные идентификаторы клиентов). Структура:
    - Node – идентификатор клиента
    - TSRs – число запросов на получения исходных данных о задаче
    - REQs – число запросов порций
    - ASWs – число возвращенных результатов

При необходимости сформировать выдачу общей информации о расчете, отличной от формата по умолчанию, можно использовать (с помощью опции -t) файл шаблона выдачи. Данный файл может иметь произвольную структуру. Для подстановки данных о расчете в файле используются метапеременные, вхождения которых будут заменяться на необходимые данные (см., например, структуру файла default_html.tpl). Допустимые метапеременные:

<%LogFileName%>	Имя анализируемого лог-файла
<%TaskName%>	Имя задачи
<%TaskArgs%>	Аргументы (входные данные) задачи
<%ServerName%>	Адрес хоста, на котором работал сервер
<%StartTime%>	Время запуска сервера
<%FirstTSRTime%>	Время получения сервером первого запроса типа TSR
<%EndTime%>	Время завершения работы сервера
<%TotalTime%>	Общее время работы сервера
<%TotalWorkingTime%>	Время работы сервера, начиная с поступления первого запроса (т.е. без учета начального ожидания клиентов)
<%OutDataFile%>	Имя файла, в который сохраняется выдача общей информации
<%Nodes%>	Число узлов, участвовавших в расчете (считаются уникальные IP-адреса)
<%Sessions%>	Число клиентских сессий

<%Processes%>	Число клиентских процессов с уникальными идентификаторами
<%ServerTime%>	Условное суммарное время сервера, потраченное на обработку каждой порции
<%ClientsTime%>	Суммарное процессорное время обработки порций на узлах
<%ServerEfficiency%>	Отношение серверного и клиентского времени – оценка эффективности работы серверной части
<%Portions%>	Общее число порций
<%TSRs%>	Число запросов типа TSR (получение сведений о задаче) к серверу
<%REQs%>	Число запросов типа REQ (получение порций) к серверу
<%ASWs%>	Число запросов типа ASW (отправка результатов) к серверу
<%ClientsEfficiency%>	Отношение числа отправленных к числу принятых порций (характеризует эффективность работы клиентов с точки зрения потерь порций)
<%MinTaskTime%>	Минимальное время обработки порции на узле
<%MaxTaskTime%>	Максимальное время обработки порции на узле
<%AverageTaskTime%>	Среднее время обработки порций на узле
<%ServerDataSum%>	Суммарный объем данных, полученных и отправленных сервером
<%ServerDataIn%>	Суммарный объем данных, полученных сервером
<%ServerDataOut%>	Суммарный объем данных, отправленных сервером
<%ClustersInfoTableHead%>	Заголовок таблицы с общими данными по каждому кластеру
<%ClustersInfoTable%>	Тело таблицы с общими данными по каждому кластеру
<%ClustersDataTableHead%>	Заголовок таблицы с данными о сетевых обменах по каждому кластеру
<%ClustersDataTable%>	Тело таблицы с данными о сетевых обменах по каждому кластеру

10. Обеспечение безопасности в системе X-Com^2

10.1. Краткое описание используемых средств

В целях защиты от анализа и/или модификация трафика в ходе распределенных расчетов, а также от подмены одной из сторон передачи данных в системе X-Com^2 реализованы механизмы идентификации клиентов и серверов, шифрования трафика и верификации результатов вычислений. Шифрование трафика серьезно затрудняет его анализ и модификацию. Идентификация при каком-либо общении клиента с сервером позволяет установить, являются ли обе стороны доверенными. Это не дает злоумышленнику их подменить или внедрить своего клиента.

Механизмы идентификации и шифрования реализованы с помощью OpenSSL – открытой реализации протоколов TLS/SSL, а также утилит для шифрования, создания центров сертификации (ЦС), проверки сертификатов и др. В клиентской части X-Com^2 используется модуль Crypt::SSLeay, обеспечивающий поддержку протокола HTTPS для LWP::UserAgent (libwww-perl). В серверной части X-Com для поддержки TLS/SSL необходимы модули Net::SSLeay, IO::Socket::SSL и HTTPS::Daemon::SSL. При подключении клиента производится SSL handshake, который параметризуется указанием сертификата и ключа сервера, а также директории с сертификатами доверенных ЦС.

Каждый работающий клиент или сервер должен использовать собственные ключи шифрования и собственный сертификат. Поскольку одно лицо может отвечать за работу сразу множества клиентов X-Com^2, было предложено использовать сертификаты, условно называемые Сертификаты Администратора клиентов, которые могут быть использованы в качестве сертификата локального ЦС, предназначенного для подписи сертификатов клиентов, находящихся в ведении Администратора. В таком случае, для того, чтобы сервер мог аутентифицировать клиента с сертификатом, подписанным администратором клиента, в директории TRUSTED на сервере должны лежать сертификаты администратора, сертификат доверенного ЦС, подписавшего сертификат администратора, и так далее до сертификата корневого ЦС.

Для того, чтобы клиент мог аутентифицировать сервер, в директории TRUSTED должны быть сертификаты, составляющую цепочку от ЦС, подписавшего сертификат сервера до корневого ЦС. Файлы CRL, изданные каждым ЦС из упомянутых цепочек, также должны присутствовать в TRUSTED, что пресечет успешную аутентификацию стороны, пользующейся не действительным сертификатом.

10.2. Создание сертификатов серверов и клиентов

Перед началом работы в режиме с поддержкой шифрования требуется создать файлы ключей и сертификатов сервера и клиентов. В дистрибутив X-Com^2 входят скрипты-оболочки OpenSSL для ведения Certification Authority в стандартной конфигурации OpenSSL (используемый файл конфигурации указан в переменной окружения $SSLEAY_CONFIG или находится обычно в */etc/ssl/openssl.cnf в Linux и OpenSSL\bin\openssl.cfg в Windows)*: rootCA.pl и adminCA.pl. Также в дистрибутиве содержится пример файла конфигурации OpenSSL openssl_example.cfg с описанием всех параметров.

Скрипт rootCA предназначен для создания корневого Certification Authority, создания подписанных файлов сертификатов серверов и сертификатов администраторов клиентов. Скрипт adminCA предназначен для создания Certification Authority администратора клиентов, выпуска сертификатов клиентов, находящихся в ведении администратора, аутентичность которого должна быть подтверждена сертификатом, выданным с использованием корневого CA.

Файл конфигурации OpenSSL задает значения, используемые по умолчанию при ведении CA, описание структуры каталогов CA (структура каталогов, описанная в файле

openssl_example.cfg необходима для корректной работы скриптов rootCA и adminCA), а также параметры и ограничения, влияющие на процесс работы пользователя с OpenSSL.

Все создаваемые сертификаты, файлы ключей, файлы запросов на выдачу сертификатов и файлы списков отозванных сертификатов хранятся в формате PEM.

При создании каждого нового сертификата пользователю предлагается ввести двухбуквенный код страны, город, название организации и подразделения, а так же CN (Common Name – имя удостоверяемой сущности или имя пользователя) и e-mail. При этом, если создается новый закрытый ключ, пользователю будет предложено ввести пароль для того, чтобы хранить и передавать ключ в зашифрованном виде. Для дальнейшего использования созданных ключей после их размещения на целевых машинах, их необходимо расшифровать. Для этого необходимо выполнить команду

```
openssl rsa -in <файл зашифрованного ключа.pem> -out <выходной файл.pem>
```

при этом пользователю предлагается ввести указанный при создании ключа пароль.

Параметры запуска для rootCA и adminCA:

-newca	Создание структуры каталогов для нового CA в текущей директории, создание и подпись корневого сертификата. При этом rootCA создает самоподписанный корневой сертификат, а adminCA создает файл-запрос на выдачу сертификата ./demoCA/careq.pem, который необходимо подписать rootCA и поместить полученный сертификат в файл ./demoCA/cacert.pem
-newcert <newkey.pem> <newcert.pem>	Создание файла закрытого ключа <newkey.pem> и подписанного файла сертификата <newcert.pem>.
-sign <request.pem> <newcert.pem>	Создание файла сертификата сервера или клиента <newcert.pem> на основе запроса на выдачу сертификата <request.pem>.
-revoke <cert.pem>	Отзыв файла сертификата <cert.pem>. Такой сертификат в базе данных CA будет числиться недействительным.
-gencrl <crlfile.pem>	Создание списка отозванных сертификатов на основе имеющихся в CA данных об отозванных сертификатах в виде файла CRL <crlfile.pem>.
Только для rootCA: -signca <careq.pem> <cacert.pem>	Создание сертификата <cacert.pem> для adminCA на основе запроса <careq.pem>.

Пример сценария использования корневого CA:

1. Поместить файл rootCA.pl в пустую директорию, которая будет в дальнейшем использоваться для хранения данных CA. Например, */home/user/MyRootCa.*
2. Из этой директории выполнить команду ***perl rootCA.pl -newca***. Далее пользователю будет предложено ввести пароль для защиты закрытого ключа CA и данные, которые будут в дальнейшем содержаться в сертификате CA. При выполнении указанной команды в директории */home/user/MyRootCa* создается необходимая для ведения CA структура каталогов:
    - /demoCA
        - /certs – *директория, в которой хранятся изданные сертификаты*
        - /crl – *директория для хранения списков отозванных сертификатов*
        - /newcerts – *альтернативная директория для хранения изданных сертификатов*

- /private – *директория, содержащая закрытые ключи*
  - cakey.pem – *файл закрытого ключа CA*
- cacert.pem – *сертификат CA*
- careq.pem – *запрос CA*
- crlnumber – *файл-счетчик номеров списков отозванных сертификатов*
- index.txt – *индексный файл-список изданных и отозванных сертификатов*
- serial – *файл-счетчик серийных номеров сертификатов*

3. При получении запроса на создание сертификата (файл формата PEM), в зависимости от предполагаемого его назначения следует выполнить команду
   `perl rootCA.pl -signca <путь к файлу запроса> <файл сертификата>`
   в случае, когда сертификат предназначен для использования в CA администратора, или
   `perl rootCA.pl -sign <путь к файлу запроса> <файл сертификата>`
   для случаев, когда такой сертификат для выпуска других сертификатов использоваться не будет (например, если сертификат требуется для аутентификации сервера X-Com). При создании сертификатов запрашивается пароль закрытого ключа, который задавался при создании корневого CA.
4. Существует возможность создавать сертификаты сервера и клиентов вместе с файлами закрытых ключей с помощью команды
   `perl rootCA.pl -newcert <путь к файлу ключа> <путь к файлу сертификата>`
   При этом пользователю необходимо будет ввести пароль для защиты закрытого ключа, а также данные полей сертификата. Однако, из соображений безопасности, использование возможности создания сертификата в паре с закрытым ключом не рекомендуется, если существует опасность кражи файлов закрытых ключей.

Пример сценария использования CA администратора клиентов:

1. Поместить файл adminCA.pl в пустую директорию, которая будет в дальнейшем использоваться для хранения данных CA. Например, */home/user/MyAdminCa*
2. Из этой директории выполнить команду **`perl adminCA.pl -newca`**. Далее пользователю будет предложено ввести пароль для защиты закрытого ключа CA и данные, которые будут в дальнейшем содержаться в сертификате CA. Необходимо учитывать, что при вводе поля CN (common name) его значение должно отличаться от CN всех других сертификатов, использующихся в работе X-Com (CN корневого CA, CA других администраторов клиентов). При выполнении указанной команды в директории */home/user/MyAdminCa* создается необходимая для ведения CA аналогичная rootCA структура каталогов.
3. Файл */home/user/MyAdminCa/demoCA/careq.pem* в дальнейшем необходимо передать владельцу корневого CA, чтобы тот создал на его основе подписанный корневым CA сертификат для CA администратора. Полученный сертификат необходимо поместить в файл */home/user/MyAdminCa/demoCA/cacert.pem*
4. Для создания сертификата клиента на основе запроса на выдачу сертификата в формате PEM необходимо выполнить команду
   `perl adminCA.pl -sign <файл запроса > <файл сертификата >.`
   При выполнении команды необходимо ввести пароль закрытого ключа администратора клиентов.
5. Существует возможность создавать сертификаты клиентов вместе с файлами закрытых ключей с помощью команды
   `perl adminCA.pl -newcert <путь к файлу ключа > <путь к файлу сертификата >`
   При этом пользователю необходимо ввести пароль для защиты закрытого ключа, а также данные полей сертификата. Использование данной команды не рекомендуется, если существует опасность кражи файлов закрытых ключей.

Отзыв и издание списков отозванных сертификатов:

1. В случае если имеется достоверная информация о краже закрытого ключа кого-либо из держателей сертификатов, изданных CA пользователя (или в случае, когда есть другие основания прекратить действие какого-либо из изданных сертификатов), необходимо выполнить команду
   `perl rootCA.pl -revoke <файл сертификата>`
   (adminCA вместо rootCA в зависимости от используемого CA). В БД CA пользователя данный сертификат будет помечен как отозванный.
2. Для издания списка отозванных сертификатов (CRL) необходимо выполнить команду
   `perl rootCA.pl -gencrl <файл CRL>`
   (adminCA вместо rootCA в зависимости от используемого CA).

10.3. Запуск серверной части X-Com

Перед запуском сервера X-Com[2] в режиме с поддержкой шифрования и аутентификации необходимо создать файл закрытого ключа сервера в формате PEM и сертификат (также в формате PEM), подтверждающий аутентичность сервера. Для этого можно воспользоваться одним из скриптов CA, описанных выше, какой-либо сторонней утилитой для ведения Certification Authority, или выполнить сначала команду

`openssl req -newkey rsa:<длина ключа, бит> -keyout <файл ключа.pem> -out <файл запроса на выдачу сертификата.pem>`

после чего получить сертификат на основе файла запроса на выдачу сертификата у доверенного Certification Authority.

При включенной аутентификации клиентов, необходимо, чтобы сертификаты доверенных CA (в том числе сертификаты администраторов клиентов) и списки отозванных сертификатов лежали в одном каталоге. Файлы должны иметь имена <x509 hash>.r0 (.r1, .r2…), где x509 hash – значение хэш-функции, вычисленной по имени держателя сертификата, которое можно получить, выполнив команду

`openssl x509 -hash -in <файл сертификата или crl> –noout`

Для получения такой структуры имен файлов достаточно поместить файлы доверенных CA в один каталог и запустить скрипт c_rehash.pl следующим образом:

`c_rehash.pl <путь к каталогу>`

Содержимое этого каталога в дальнейшем используется сервером X-Com для проверки сертификатов клиентов.

Параметры сервера для работы с поддержкой шифрования и аутентификации клиентов (указываются в .ini-файле):

useSSL	Режим шифрования: noverify – режим с поддержкой шифрования и аутентификации сервера. Аутентификация клиентов не производится, поэтому указание параметра SSLdir не требуется. verify – шифрование и аутентификация клиентов и сервера включена. verifycrl – шифрование и аутентификация клиентов и сервера включена, при этом дополнительно включается проверка всех сертификатов по CRL. (Для корректной работы в таком режиме необходимо, чтобы установленная версия библиотек OpenSSL была не ниже 0.9.7b) Все другие значения или отсутствие указание параметра useSSL подразумевают отключение режима шифрования и аутентификации сервера и клиентов. При этом значения параметров SSLkey, SSLcert и SSLdir игнорируются и могут быть опущены.
SSLkey	Полный путь к файлу закрытого ключа сервера.
SSLcert	Полный путь к файлу сертификата сервера.

SSLdir	Полный путь к каталогу доверенных сертификатов и списков отозванных сертификатов.

При включенном шифровании сервер использует для общения с клиентами протокол https.

10.4. Запуск клиента X-Com^2

Перед запуском клиента X-Com^2 также необходимо иметь в наличии файлы ключа и сертификата клиента, а также поместить сертификаты доверенных CA и CRL в отдельный каталог, переименовав файлы с использованием скрипта c_rehash.pl.

Параметры командной строки для запуска клиента с поддержкой шифрования и аутентификации сервера и клиентов:

--key-file	Полный путь к файлу закрытого ключа клиента.
--cert-file	Полный путь к файлу сертификата клиента.
--ca-path	Полный путь к каталогу доверенных сертификатов и списков отозванных сертификатов.

При включенном шифровании, в URL сервера X-Com^2 в качестве протокола указывается https вместо http.

Пример запуска клиента без шифрования:

```
perl -T client.pl -s http://localhost:65002/
```

Пример запуска клиента с поддержкой шифрования и аутентификации сервера:

```
perl -T client.pl -s https://localhost:65002/ --ca-path /home/user/certs/trusted
```

Пример запуска клиента с поддержкой шифрования и аутентификации клиентов и сервера:

```
perl -T client.pl -s https://localhost:65002/ --ca-path /home/user/certs/trusted
--key-file /home/user/clientkey.pem --cert-file /home/user/clientcert.pem
```