1 Система управления локальной дисковой памятью

1.1 Общее описание

Система предназначена для сбора, хранения и анализа информации о состоянии локальных дисков параллельной вычислительной системы с целью обеспечить возможность работы с группами файлов, представляющих состояния вычислительных процессов.

Основными сущностями в системе управления ЛДП являются: диск — локальная файловая система некоторого вычислительного узла, файл, задача и контрольная точка — группа файлов, ассоциированная с задачей. Создаваемые по мере работы приложения контрольные точки регистрируются в системе, после чего происходит планирование операций создания избыточной информации для предотвращения возможных сбоев.

Благодаря тому, что вся информация находится в файловом индексе, можно обеспечить выполнение операций восстановления в случае сбоев или временной недоступности вычислительных узлов, проверку целостности файлов и их групп, а также автоматическое удаление устаревшей информации.

Основной менеджер системы реализован с использованием языка высокого уровня Haskell, что позволяет обеспечить удобное изменение алгоритмов поддержания целостности и легкость построения и проверки математических моделей.

1.1.1 Структура индекса

Файловый индекс состоит из переферийных демонов и центрального процесса. Последний представляет из себя нить ввода-вывода, написаную на C++, и собственно ядро на Haskell. Предусмотрен режим без IO нити, однако его функционал весьма ограничен. Также, к основному процессу подключаются вычислительные приложения, передающие информацию о созданных ими файлах, например, используя библиотеку `libccp2` (1.2)

Сервер ввода-вывода Представляет из себя отдельную нить с запущеным сетевым сервером на TCP портах 2563 и 2564 и считывающий команды из стандартного ввода. На 2563 порту принимаются обычные нешифрованные соединения. На 2564 — принимаются SSLv3 соединения

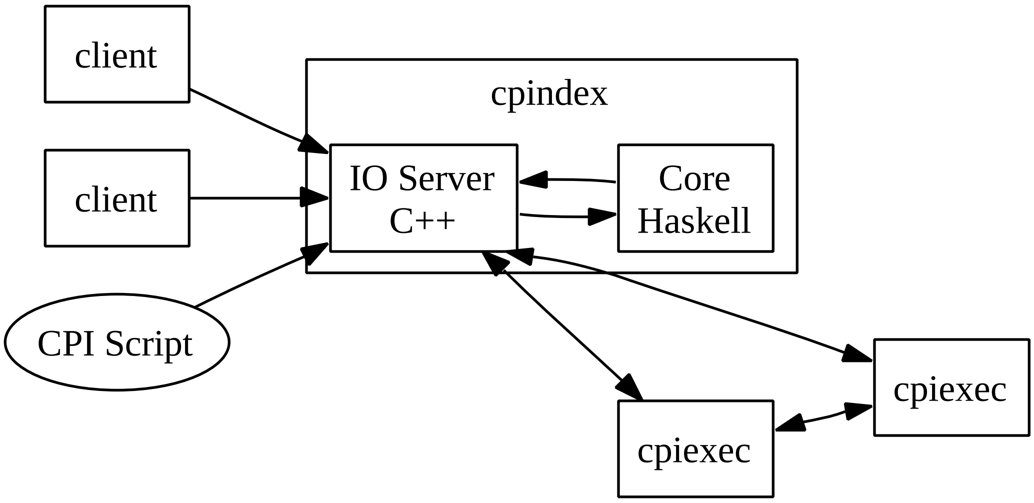

Рис. 1: Структура индекса.

с шифрованием `RC4-SHA` и проверкой сертификата клиента. Если проверка не проходит, соединение обрывается, таким образом есть возможность ограничить доступ к индексу лишь авторизованым приложениям или интегрировать в систему Globus. На данный момент есть некоторые проблемы с обработкой прокси-сертификатов задач, но их решение не предоставляет технической трудности и будет решено при внедрении системы.

На данный момент не реализована система разграничения прав в связи с тем, что система еще не подошла к стадии внедрения и простота отладки ставится выше необходимости оградить систему от несанкционированного доступа.

Дисковые демоны `cpiexec` Подбробно они описаны в 1.1.2

Скрипты Команды могут считываться не только из сетевого соединения, но так же из локальных файлов. Основное применение — загрузка базы при старте. По умолчанию используется файл `save.cpidb`.

1.1.2 Дисковые демоны

Кроме центрального индекса созданы специальные дисковые демоны, обеспечивающие низкоуровневые операции с файлами, такие как хеширование или удаление. На данный момент реализованы некоторые основные команды, такие как удаление файла, подсчет и проверка его контрольной суммы (MD5). Кроме этого, на дисковый демон возлагается задача создания избыточной информации и перемещения файлов между узлами.

При запуске он соединяется с индексом и передает ему информацию о диске, на котором запущен. В случае, если в индексе этого диска не было — он добавляется, иначе информация о нем обновляется. Идентификатор диска создается методом из библиотеки libccp2 (см. 1.2).

Дисковый демон хранит список всех файлов, заведенных в индексе для данного диска — при старте он запрашивает у индекса текущий список; после этого регистрируется перенаправление для данного диска и при добавлении файла информация о нем поступает в демон. При выполнении операции, тербующей обновления информации в индексе, такой как вычисление контрольной суммы, посылается соответсвующая команда.

Создание избыточной информации происходит следующим образом. Сначала на всех демонах инициализируется операция с уникальным идентификатором. После этого для каждого процесса, учавствующего в построении коммуникационной сети, передается набор команд по выстраиванию его локальных связей между входными и выходными потоками и указания, с какими надо соединяться по сети или какие соединения принмать (подробнее о построение коммуникационных сетей написано в **??**). Если в процессе построения произойдет сбой, данная незаконченая операция будет удалена на всех демонах. После этого, всем процессам высылается команда запустить операцию. Не учавствующе при этом удалят у себя информацию о ней как ненужную. При таком подходе устраняется опасность возникновения записей, не относящихся ни к одной запущеной или еще готовящейся операции. При этом логика построения сетей вынесена в индекс, где могут быть реализованы рассмотреные в **??** алгоритмы.

На данный момент возможно построение любых сетей с использованием файлов, TCP-соединений и вычисления XOR-сумм, этого достаточно для обеспечения отказоустойчивости по схеме XOR (см. **??**)

1.1.3 Протокол взаимодействия с файловым индексом

Существует два класса команд — те, что напрямую передаются в ядро индекса и те, которые обрабатываются в С-коде. Если команда начинается с символа /, то она разбирается согласно правилам второго класса, иначе — первого. Первый класс не предназначен для использования вручную. Он имеет формат `«команда» «тип данных» «данные»`, где `«данные»` представляют из себя строковое представление Haskell типов. Тип данных может быть одним из следующих

- `ActCP` данные контрольной точки
- `ActFile` описание файла
- `ActDisk` информация диска
- `ActReplica` файловой реплики
- `ActCmd` строковой параметр команды

Нижу приводятся стандартные команды с описанием их действия и разбиению по группам.

Команды, частично обрабатываемые в серверной части Эти команды не поступают напрямую в ядро индекса, а обрабатываются в С-сервере.

- /cntl/alias «имя соединения» Установить для данного соединения дополнительное символьное имя, подбробнее в **??**.
- /cntl/load «имя файла» Загрузить базу из указаного файла
- /cntl/save «имя файла» Сохранить текущую базу в указаный файл
- /cntl/close Закрыть текущее соединение. Используется для закрытия файлов, запись в которые завершена. Игнорируется для стандартного ввода.
- /cntl/shutdown Выставить флаг остановки, при закрытии файла будет послана команда halt. Используется командой shutdown

Контрольные команды. Команды, используемые для управлением работой файлового индекса, такие как отключение сетевого сервера, отключение клиента или загрузки другой базы.

- halt Остановить сервер без сохранения базы.
- shutdown Остановить сервер, предварительно сохранив базу в файл save.cpidb
- start Запустить сетевой сервер. Имеет смысл лишь при запуске индекса в безсетевом режиме.
- stop Остановить сетевой сервер. Считывание из стадартного ввода будет производится в упрощенном режиме.
- save Выдать в поток список команд для создания базы в ее текущем состоянии. Поток после этого закрывается.
- load:test Загрузить тестовую базу.

Команды приведения индекса к целостному виду.

- rebuild Проверить и пересоздать списки ссылок на файлы в контрольных точках и в файлах на точки.
- consistent Удалить из базы ненужную информацию. Будут удалена следующая информация:
  - файлы, ссылающиеся на несуществующий диск.
  - файлы, которые не принадлежат ни одной контрольной точке
  - точки, не содержащие ни один файл
  - точки, для которых невозможно восстановление

1.1.4 Состояния системы

Состояние файлового индекса $s \in \mathbb{S}$, где

$$\mathbb{S} \subset \mathbb{S}_C \times \mathbb{S}_F \times \mathbb{S}_D \times \mathbb{S}_R,$$

определяется текущим состоянием дисков ($\mathbb{S}_D$), файлов ($\mathbb{S}_F$), контрольных точек ($\mathbb{S}_C$) и файловых реплик ($\mathbb{S}_R$). При этом, текущее состояние дисков $s_d \in \mathbb{S}_D$ ограничивает возможное пространство состояний файлов $\hat{\mathbb{S}}_F|_{s_d} = \hat{\mathbb{S}}_F(s_d) \subset \mathbb{S}_F$, аналогично для s_f и $\hat{\mathbb{S}}_R$, s_r и $\hat{\mathbb{S}}_C$. То есть,

$$s \in \mathbb{S} \equiv \{s_c, s_f, s_d, s_r | s_c \in \hat{\mathbb{S}}_C|_{s_r}, s_r \in \hat{\mathbb{S}}_R|_{s_f}, s_f \in \hat{\mathbb{S}}_F|_{s_d}\}$$

Подробнее зависимости $\hat{\mathbb{S}}_F$, $\hat{\mathbb{S}}_R$ и $\hat{\mathbb{S}}_C$ описаны в 1.1.6, 1.1.7 и 1.1.8 соответсвенно.

При реализации были использованы следующие языковые конструкции

$\mathbf{type}\ \mathit{CPIDB} = ([\,\mathit{CPInfo}], [\,\mathit{CPFile}], [\,\mathit{CPDisk}], \mathit{ReplicaList})$

1.1.5 Состояния дисков

"Диском" в системе называется какое-то хранилище файлов с уникальным идентификатором, которое замонтировано хотя бы на одном узле. Соответсвенно, диск описывается своим идентификатором, сетевым адресом и точкой монтировани. Кроме этого возможно одно из следующих состояний:

- доступен и работает без ошибок
- отключен на какое-то время
- вышел из строя
- на диске запущена проверка целостности

$\mathbf{data}\ \mathit{DiskState} = \mathit{DiskOk} \mid \mathit{DiskNotAvail} \mid \mathit{DiskDead} \mid \mathit{DiskCheck}\ \mathit{String}$
$\quad \mathbf{deriving}\ (\mathit{Eq}, \mathit{Ord}, \mathit{Show}, \mathit{Read})$

Соответсвенно, изменение состояние диска может вызвать изменение состояния всех файлов, на нем находящихся.

1.1.6 Состояния файлов

Файл описывается следующими полями:

- уникальный идентификатор файла
- физическое расположение файла
  - идентификатор диска
  - путь к файлу
  - контрольная сумма файла
- ссылки на контрольные точки, которым этот файл принадлежит

  – идентификатор точки
  – номер в наборе
  – последовательный номер файла

Файл может находится в одном из следующих состояний:

- в порядке; последняя проверка контрольной суммы дала положительный результат, диск, на котором находится файл в порядке
- поврежден; файл не найден или последняя проверка контрольной суммы показала что файл изменился
- недоступен; диск, на котором хранится файл в данный момент недоступен
- считается контрольная сумма файла
- не готов; например, при копировании файла с узла на узел
- восстанавливается;

  data *FileState* = *FileOk* | *FileCorrupt* | *FileNotAvail* | *FileHashing* |
  FileNotReady | *FileRecovery String*
  deriving (*Eq*, *Ord*, *Show*, *Read*)

  Соответсвенно, существуют следующие ограничения $\hat{\mathbb{S}}_F|_{s_d}$:

- *DiskNotAvail*: все файлы, находящиеся на этом диске, могут быть лишь в состояниях *FileCorrupt* или *FileNotAvail*
- *DiskDead*: файлы должны быть в состоянии *FileCorrupt*

1.1.7 Состояния файловых реплик

Подробно файловые реплики описаны в 1.1.9. Реплика может находится в одном из следующих состояний:

- все файлы в порядке
- часть файлов повреждена или недоступна
- все файлы либо повреждены либо недоступны

  data *ReplState* = *ReplOk* | *ReplCorrupt* | *ReplDead*
  deriving (*Eq*, *Ord*, *Show*, *Read*)

1.1.8 Состояния контрольных точек

Контрольной точкой считается некоторый набор файлов, имеющий ценность только когда все файлы доступны и не повреждены. Она описывается следующими полями

- идентификатор точки
- идентификатор задачи, которая создала данную точку; необходим для связи разных точек одной задачи и управления ими
- схема обеспечени отказоустойчивости; возможны следующие варианты:

  plain избыточной информации нет; любой сбой приводит точку в невосстановимое состояние

  xor отказоустойчивость обеспечивается дополнительным файлом с xor-суммой; возможно восстановление после единичного сбоя

- ссылки на файловые реплики, аналогичные описаным в 1.1.6, но с идентификатором точки замененным на идентификатор реплики

Для точки определены следующие состояния

- все файлы доступны и не повреждены
- восстановимый сбой; часть файлов либо недоступна, либо повреждена
- невосстановимый сбой; данная точка должна быть вычищена из системы
- точка в процессе сборки; в данный момент поддерживается только сборка синхронных точек с фиксированным последовательным номером файла
- точка в процессе восстановления после сбоя

$$\mathbf{data}\ \mathit{CPState} = \mathit{CPOk} \mid \mathit{CPFailed} \mid \mathit{CPDead} \mid \mathit{CPAssemble}\ (\mathit{Int}, \mathit{Int}) \mid \mathit{CPRecovery}\ \mathit{String}$$
$$\mathbf{deriving}\ (\mathit{Eq}, \mathit{Ord}, \mathit{Show}, \mathit{Read})$$

1.1.9 Файловые реплики

Контроль реплик позволяет сохранять информацию об идентичности файлов при копировании или производить поиск файловых дублей. Это дает возможность восстановить файл менее затратными средствами, если где-то сохранилась его копия.

Реплика — это отображение

$$r : \mathbb{S} \times \mathbb{U}_l \to \{u_{f1}, u_{f2}, \ldots, u_{fk} | k \geq 0, u_{fi} \in \mathbb{U}_f\}$$

где $\mathbb{S}$ — состояния системы (см. 1.1.4), $\mathbb{U}_l$ — пространство логических идентификаторов, uuid наборов идентичных файлов, на которые могут ссылаться контрольные точки, а $\mathbb{U}_f$ — uuid физических файлов (см. **??**) Ниже мы будем рассматривать ограничение на текущее состояние системы s отображения $r|_s \equiv r_s$. Поскольку пространства имен логических и физических идентификаторов различны, требование на глобальную уникальность в $\mathbb{U}_l \cup \mathbb{U}_f$ можно ослабить до уникальности отдельно в $\mathbb{U}_l$ и в $\mathbb{U}_f$.

Наложим на r_s следующие ограничения:

$$r_s(u_l) = (u_{f1}, \ldots, u_{fk}), \text{то } \forall i, r_s(u_{fi}) \equiv r_s(u_l)$$

То есть, физические идентификаторы взаимозаменяемы. Но не верно, что

$$r_s(\hat{u}_l) \equiv r_s(u_l) \Rightarrow \exists j\, \hat{u}_l = u_{fj}$$

Например, если один из объектов, физический идентификатор которого находился в наборе, был поврежден и удален из системы.

При реализации использованы следующие языковые конструкции

type $\mathit{Replica} = (\mathbb{U}_l, [\mathbb{U}_f], \mathit{ReplState})$

type $\mathit{ReplicaList} = [\mathit{Replica}]$

$\mathit{replica} :: \mathit{ReplicaList} \to \mathbb{U}_l \to [\mathbb{U}_f]$

Где *ReplicaList* — список, в котором хранятся записи о файловых репликах, а *replica* определяет отображение r_s в зависимости от текущего списка. При этом, если для запрашиваемого логического идентификатора нет записей в списке реплик, определим $r_s(u_l) = \{u_f | u_f \equiv u_l\}$ Таким образом для файлов, не имеющих копий, можно не заводить запись в списке, также, можно удалятять записи относящиеся к невосстановимым наборам.

1.2 Библиотека оповещения о создании контрольной точки libccp2

Данная библиотека позволяет вычислительному приложению оповещать систему обеспечения отказоустойчивости о создании координированной контрольной точки. При этом на файловый индекс cpi TCP/IP передается вся необходимая информация о координированной контрольной точке. libccp2 является переработкой описаной в предыдущем отчете прикладной библиотеки и практически полностью совместима с ней по API. Ниже приведен краткий список изменений:

- изменен протокол передачи информации демону с двоичного на текстовый, описаный в 1.1.3
- добавлены поля UUID для файлов и контрольных точек
- появились вызовы для более простой сборки синхронных контрольных точек (`ccpc_init_sync` и `ccpc_register_sync`)
- добавлена возможность устанавливать защищеные соединения с проверкой подлинности по X.509 сертификатам

Оповещение сервера происходит следующим образом. Устанавливается TCP/IP соединение с файловым индексом и на него отправляются команды создания контрольной точки и набора ее файлов. При этом передается

- uuid точки, если установлен флаг `CCPC_FLAGS_UUID_GEN` — генерируется при отправке, иначе — нули
- uuid задачи, считывается из переменной окружения `TASK_UUID` при старте вычислительного процесса.
- схема обеспечения отказоустойчивости выставляется в plain, то есть без избыточной информации.

Далее для каждого файла передается

- абсолютное имя файла
- uuid узла или файловой системы, по которому можно определить расположение файла

- идентификатор контрольной точки, для связи в индексе
- номер файла в наборе
- последовательный номер файла контрольной точки

1.2.1 Пользовательский интерфейс

Пользовательский интерфейс библиотеки состоит из вызовов инициализации, непосредственно передачи сообщений и вызова завершения работы. Ниже они описаны более подробно

`ccpc_flags_set(int flags)`

Устанавливает различные флаги, так или иначе изменяющие поведение библиотечных вызовов. В качестве параметра возможно логическое ИЛИ следующих флагов:

> `CCPC_FLAGS_DEBUG`
>
> Активировать дополнительные отладочные распечатки о посылаемой на сервер информации.
>
> `CCPC_FLAGS_NODENAME`
>
> Брать в качестве UUID файловой системы первые 16 символов имени узла, получаемого через uname
>
> `CCPC_FLAGS_ENVUUID`
>
> Брать в качестве UUID приложения значение из переменной окружения с именем TASK_UUID. В случае если она содержит некорректно отформатированное значение будет выдано сообщение об ошибке и идентификатор приложения будет взят нулевым.
>
> `CCPC_FLAGS_SEQ_ID`
>
> В случае, если выставлен этот флаг, на сервер будут отсылаться не имена файлов, а номера контрольных точек для соответствующих узлов. Имя файла будет создано только для локальной точки. Подробнее об этом в описании функции `ccpc_register`.

`CCPC_FLAGS_UUID_GEN`

Создавать уникальные идентификаторы файлов и контрольных точек при их отправке. Считываются из /proc/sys/kernel/random/uuid. В будущем, возможно, будет заменено на использование вызовов из системной библиотеки libuuid

`ccpc_flags_unset(int flags)`

Аналогично функции ccpc_flags_set, только снимает флаги, указанные в качестве аргумента.

`ccpc_init(int numproc, int templ_size, const char * templ)`

Заполняет внутренние структуры информацией о файловой системе и идентификаторе приложения, а также инициализирует шаблоны имен файлов контрольных точек. Шаблон может содержать две подстановки целочисленных переменных в формате описанном для библиотечной функции printf. Первая будет заменена на номер узла для которого создается файл контрольной точки, вторая — на номер точки. Пример: "e2d.%02d-%04d.ccp". Данный вызов должен быть произведен до любого применения ccpc_register. Перед ним возможно вызывать только ccpc_flags_set или ccpc_flags_unset

`ccpc_init_sync(int numproc, int local_rank)`

Вызов аналогичный паре `ccpc_init` и `ccpc_rank_local` для последующего использовния сборки синхронных точек вызовом `ccpc_register_sync`. Шаблон имен файлов не используется. Вызовы `ccpc_register` не будут корректно работать.

`ccpc_init_serv(const char * serv_addr)`

Устанавливает IP адрес сервера, получающего сообщения. Возможно любое корректное значение в стандартном формате.

`ccpc_uuid_get_fs ( char * uuid )`

Возвращает значение полученное для UUID локальной файловой системы. В данный момент это либо нули, либо первые 16 символов имени узла полученного через вызов uname. Это зависит от того, выставлен ли флаг CCPC_FLAGS_NODENAME. Аргумент должен ссылаться на область памяти больше 16 байт, значение UUID будет возвращено в двоичном виде а не в печатном формате.

`ccpc_uuid_set( int rank, const char * uuid )`

Устанавливает значение UUID файловой системы для узла с идентификатором rank. Используется впоследствии при заполнении данных о файле контрольной точке, не используется при установленном флаге CCPC_FLAGS_SEQ_ID. Значение uuid — 16 байт двоичной информации, а не UUID в стандартном формате. Может быть использован с вызовом ccpc_uuid_get_fs для обмена информацией о UUID файловых систем на различных узлах.

`ccpc_rank_local(int rank)`

Устанавливает идентификатор локального узла, используется только при выставленном флаге CCPC_FLAGS_SEQ_ID.

`ccpc_register(const int * seq_id, int size)`

Отправить на сервер информацию о наборе файлов для контрольной точки. В параметре seq_id передается массив последовательных номеров файлов локальных точек на момент, когда был создан файл на данном узле. При этом будут произведены подстановки в шаблон, заданный вызову ccpc_init и полученные имена отправлены на сервер. В случае, если выставлен флаг CCPC_FLAGS_SEQ_ID, подстановка будет проведена только для локального узла, номер которого должен быть выставлен вызовом ccpc_rank_local. Для остальных будет отправлено не имя файла, а только номер точки.

`ccpc_register_sync(int seq_id, const char * filename)`

Зарегестрировать указанный файл как часть синхронной контрольной точки с последовательным номером `seq_id`

`ccpc_terminate()`

Отсылает на сервер данные о последней точке, прошедшей вызов ccpc_register. При этом, устанавливаются флаги, указывающие о том, что это последнее сообщение от данного процесса. После этого не должно быть вызовов ccpc_register, только ccpc_fini. Данная функция имеет смысл лишь при отсутствии флага CCPC_FLAGS_SEQ_ID и если вся информация отсылается одним выделенным процессом задания. В libccp2 данный вызов оставлен для обеспечения обартной совместимости, однако выполняемые им действия особого смысла не имеют.

`ccpc_fini()`

Освобождает все ресурсы, занятые вызовами данной библиотеки.