DOI dx.doi.org/10.24866/VVSU/2073-3984/2019-4/207-218

УДК 519.687, 004.457

Д.В. Леонтьев[1]

Д.И. Харитонов[2]

Д.С. Одякова[3]

Р.В. Парахин[4]

Институт автоматики и процессов управления Дальневосточного отделения
Российской академии наук
Владивосток. Россия

# Автоматизация постобработки данных вычислительных экспериментов*

В данной работе рассматриваются принципы работы системы событийного управления обработкой данных вычислительных экспериментов. Рассматривается построение моделей обработки данных вычислительных экспериментов. Для построения моделей используются сети Петри. Модель вычислительного эксперимента состоит из моделей вычислительного и управляющего процессов. Построение этих моделей происходит раздельно. Построение модели вычислительного процесса осуществляется в два этапа. Сначала пользователь формирует дерево событий вычислительного эксперимента, далее происходит автоматическое построение модели вычислительного процесса из дерева событий. Модель управляющего процесса строится на основе шаблонов реакции. Представлены три шаблона реакции: реакция на предыдущее событие, реакция на каждое N-е событие, реакция на следующее событие. Шаблон реакции настраивается на событие, на которое он должен реагировать. Подход позволяет строить модели обработки данных пользователям с минимальной подготовкой. Рассматривается архитектура подсистемы событийного управления вычислительным экспериментом. Приводится описание используемых инструментов (Slurm и Audit), которые являются основой для функционирования системы событийного управления. Отслеживание событий происходит с помощью подсистемы Audit, которая собирает события и передает их на узел обра-

[1] Леонтьев Денис Васильевич – научный сотрудник; e-mail: devozh@dvo.ru

[2] Харитонов Дмитрий Иванович – канд. техн. наук, ст. науч. сотрудник; e-mail: demiurg@dvo.ru

[3] Одякова Дарья Сергеевна – старший инженер-программист; e-mail: darlene@dvo.ru

[4] Парахин Роман Валерьевич – инженер-программист; e-mail: fadak@dvo.ru

* *Работа выполнена при финансовой поддержке программы «Приоритетные научные исследования в интересах комплексного развития Дальневосточного отделения РАН» (проект 18-5-78) и в рамках темы госбюджетного задания №АААА-А17-117040450019-8.*

ботки. На узле обработки расположен процесс управления, отслеживающий события на модели и запускающий выполнение соответствующих реакций. Приводится описание процесса запуска вычислительного эксперимента с событийным управлением. Описывается алгоритм свертки последовательности событий, предназначенный для поиска несоответствий между моделью и реальным вычислительным экспериментом. Важной чертой подхода является отсутствие необходимости в перепрограммировании исходной вычислительной задачи.

**Ключевые слова и словосочетания:** суперкомпьютерные вычисления, обработка больших данных, многопроцессорные вычислительные системы, визуализация научных данных, сети Петри.

D.V. Leontiev

D.I. Kharitonov

D.S. Odyakova

R.V. Parakhin

IAC PFEBRAS
Vladivostok. Russia

## Automating computational experiments data post-processing

This article considers the principles of the event control system for processing data from computational experiments. An approach to construct a data processing models of a computational experiments is considered. To make models the Petri nets are used. The model of computational experiment consists of computational and control processes models. The models are built separately. The computational process model is built in a two stages. On the first stage the user generates the event tree of computational experiment. On the second stage the computational process model is automatically built from the event tree. The model of the control process is built from a reaction patterns. The following three reaction patterns are developed: reaction on a previous event, reaction on each N-th event, reaction on a next event. The reaction pattern is configured on the triggered event. The approach allows users with minimal skills to make the data processing models. The architecture of the event control subsystem of a computational experiment is considered. A description of the tools used (Slurm and Audit), which are the basis for the functioning of the event management system, is given. Event control is performed using the Audit subsystem, which collects events and sends them to the processing node. A control process is located on the processing node, which track events on the model and starts the execution of the corresponding reactions. A description of the starting process computational experiment with event control is given. The sequence of events convolution algorithm is described, which is designed to search for inconsistencies between model and a real computational experiment. The main feature of the developed approach is that there is no need to reprogram an original computational task.

**Keywords:** high performance computing, big data processing, multiprocessor computing systems, scientific data visualization, Petri nets.

С каждым годом количество экспериментов, проводимых с помощью вычислительных кластеров, неуклонно растёт. Этому способствуют такие факторы, как дешевизна проведения эксперимента, скорость получения результатов, возможность проводить эксперименты параллельно и т.д.

Вычислительные кластеры предназначены для одновременного выполнения множества экспериментов разных пользователей, поэтому важно использовать доступные ресурсы эффективно. В процессе проведения эксперимента возможны случаи, когда расчёт расходится или возникают некоторые исключительные ситуации, при которых дальнейшее проведение эксперимента становится бессмысленным. Отслеживая такие ситуации и выполняя их обработку, можно увеличить эффективность использования оборудования. Отслеживать можно и другие события, такие, как запись в файл, запрос данных, запись в базу данных, взаимодействие с сокетами и т.д. При наличии информации о происходящих событиях появляется возможность строить пользовательские схемы обработки данных вычислительных экспериментов. Примером может служить визуализация данных эксперимента. Визуализация данных может требовать значительных вычислительных ресурсов и дискового пространства, поэтому наиболее выгодным вариантом становится визуализация данных непосредственно на кластере [1, 2], которая происходит параллельно процессу выполнения эксперимента. Такой вариант позволяет сократить время ожидания получения результатов и избавиться от загрузки результатов «неудачных» экспериментов на компьютер пользователя, что существенно экономит время. Дополнительным плюсом при визуализации данных является разгрузка сетевых соединений и как следствие более быстрая доставка результатов до конечного потребителя. Так, в процессе визуализации данные, занимающие гигабайты дискового пространства, могут быть преобразованы к читаемому пользователем виду, занимающему в сотни, а то и в тысячи раз меньше места.

Многие исследователи ставят задачи, которые необходимо автоматизировать, например, визуализация данных (при известных параметрах визуализации), выполнение анализа данных и т.д. Например, если вычислительный эксперимент позволяет, можно начать выполнять некоторую обработку данных ещё до окончания самого эксперимента. Часто для выполнения таких задач исследователям приходится разрабатывать собственные скрипты и утилиты, которые решают такие задачи. При выполнении больших экспериментов требуется использовать вычислительные кластеры. Однако не все исследователи обладают достаточной квалификацией и временем для решения задач автоматизации на кластере.

В настоящей статье рассматриваются принципы работы системы событийного управления обработкой данных вычислительных экспериментов.

**Модель вычислительного эксперимента**. Любой вычислительный эксперимент выполняется по некоторому «плану», который известен пользователю. Такой «план» включает в себя все процессы, которые происходят в рамках этого вычислительного эксперимента. Управление обработкой данных невозможно без понимания этих процессов, поэтому пользователь должен рассказать о них системе управления посредством модели вычислительного эксперимента. Модель состоит из модели вычислительного процесса и модели управляющего процесса.

Модель вычислительного процесса представляет собой сеть Петри, состоящую из множества мест и множества переходов, входной функции инцидентности переходов (мультимножества мест, необходимых для возбуждения переходов) и выходной функции инцидентности переходов. В связи с тем, что не у всех пользователей есть необходимая квалификация, построение модели происходит через промежуточное представление – дерево событий. При описании модели используются понятия, имеющие аналогии в программировании, такие, как варианты (ветвления), циклы, стадии (последовательность команд), события (команды). Дерево событий автоматически преобразуется в модель в терминах сетей Петри путём сопоставления элементов дерева определённым шаблонным конструкциям и последующей «склейки» в модель [3; 4].

Модель управляющего процесса описывает работу процесса, отвечающего за отслеживание событий, генерируемых вычислительным процессом, и выполнение реакций на эти события. Модель представляет собой сеть Петри. После построения модели вычислительного процесса каждому событию (группе событий) сопоставляется шаблон реакции. Шаблон реакции позволяет установить зависимость происходящих событий и реакции на них. В рамках данной работы пользователю предложены три шаблона реакций:

– «реакция на предыдущее событие» – шаблон предназначен для реагирования на событие, которое предшествовало текущему событию;

– «реакция на каждое N-е событие» – шаблон предназначен для реагирования на каждое N-е событие;

– «реакция на следующее событие» – шаблон предназначен для реагирования на событие, которое следует сразу за текущим событием.

В качестве примера вычислительного эксперимента используется моделирование волн цунами. Вычислительный эксперимент носит итерационный характер. Итерации происходят по расчётному времени, отличаясь друг от друга по значению этого времени. Раз в заданный временной интервал происходит запись расчётных массивов в файл с уникальным именем. Раз в заданный интервал модельного времени происходит запись усреднённых величин в лог. После записи усреднённых величин необходимо визуализировать расчётные массивы (данные берутся из файла, записанного до записи лога). По окончанию расчёта массивы данных сохраняются в файлы и записывается информация в лог, после этого происходит визуализация всех записанных массивов и генерация видеоролика.

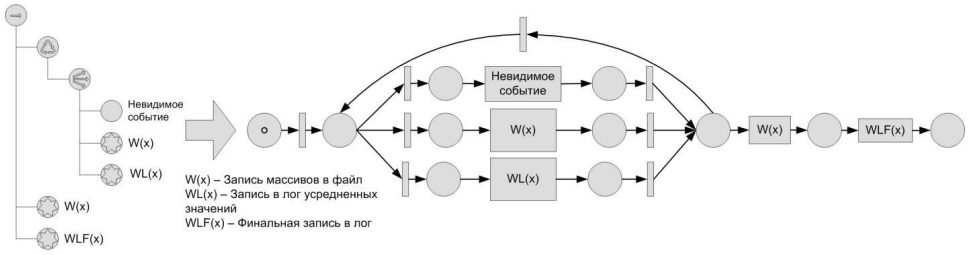


Рис. 1. Модель вычислительного процесса

Модель вычислительного процесса для данного примера представлена на рис. 1. В левой части рисунка обозначено дерево событий, которое затем автоматическим способом преобразуется в сеть Петри (справа).

Модель управляющего процесса показана на рис. 2. Согласно условиям задачи существует две реакции на события: «визуализация массива данных» и «визуализация всех массивов и генерация видеоролика». Для каждой реакции строится отдельная модель. Обе реакции соответствуют шаблону «реакция на предыдущее событие». Поэтому модель управляющего процесса в данном случае состоит из двух моделей, каждая из которых отслеживает свои события и выполняет соответствующие им реакции.

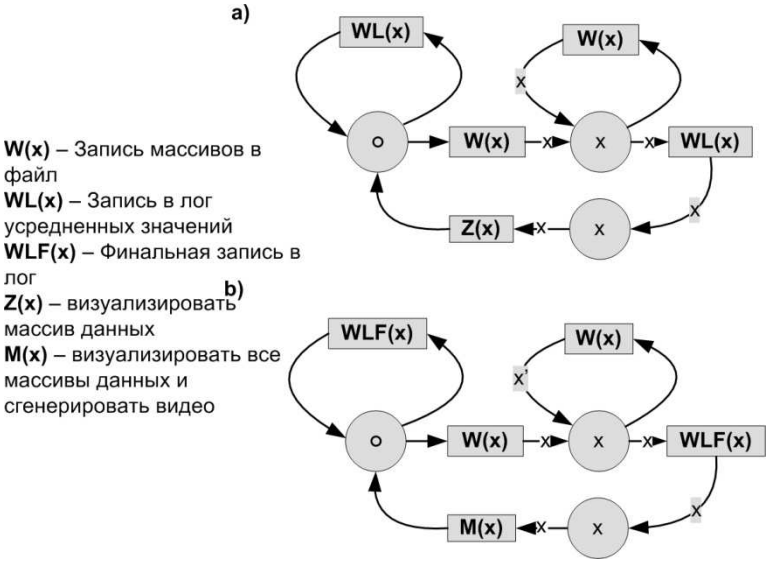


Рис. 2. Модель управляющего процесса: a) реакция на событие «запись усреднённых значений в лог», b) – реакция на событие «финальная запись в лог»

На рис. 2a приведена модель реакции на событие «запись усреднённых значений в лог, происходящая в вычислительном эксперименте в цикле». Другими словами, в процессе вычислительного эксперимента происходит получение события W(x). Параметры данного события сохраняются и передаются далее. Получение события W(x) происходит до тех пор, пока не будет получено событие WL(x), после которого будет выполнена процедура визуализации последнего массива данных Z(x), полученного в событие W(x). А на рисунке 2b представлена реакция на событие «финальная запись в лог». Отличие данной реакции от предыдущей заключается в отслеживаемом событии WLF(x) (финальная запись в лог) и реакции на него M(x) (визуализация всех массивов данных и генерация видеоролика).

**Архитектура подсистемы событийного управления**. Подсистема событийного управления предназначена для работы на вычислительных кластерах центра коллективного пользования «Дальневосточный вычислительный ресурс»

(ЦКП «ДВВР»), для управления ресурсами которого применяется система планирования заданий Slurm. Подсистема не использует уникальные для ЦКП «ДВВР» инструменты, поэтому может быть свободно использована на кластерах, применяющих такую же систему планирования. Для кластеров с альтернативными системами планирования использование подсистемы управления будет возможно после незначительных модификаций.

Slurm Workload Manager – свободный, с открытым исходным кодом планировщик заданий для Linux и Unix-подобных операционных систем, используемых многими в мире суперкомпьютерами и вычислительными кластерами. Slurm поддерживает скрипты пролога и эпилога. Скрипт пролога обеспечивает выполнение некоторых действий до запуска пользовательского задания, а скрипт эпилога после выполнения задания. На рисунке 3 представлена упрощенная схема запуска вычислительного задания. Как видно из рис.3, скриптов пролога и эпилога может быть несколько, причём скрипты пролога и эпилога, выполняемые на управляющем и вычислительных узлах, различны. По умолчанию скрипты пролога и эпилога не выполняются. Выполнение и содержимое скриптов может быть настроено администратором. В рамках данной работы используются только скрипты пролога и эпилога, которые выполняются на вычислительном узле.

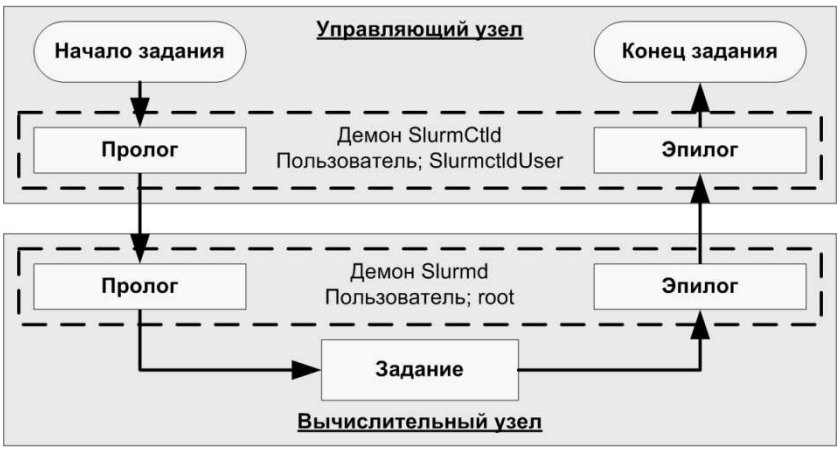


Рис. 3. Схема запуска вычислительного задания в Slurm

Основой функционирования подсистемы событийного управления является получение информации о событиях, которые происходят при выполнении заданий. Наиболее распространённый и гибкий инструмент, решающий эту задачу в операционных системах семейства Linux, – подсистема Audit.

Подсистема Audit включена в ядро Linux начиная с версии 2.6 и предназначена для проведения аудита событий операционной системы. Подсистема Audit присутствует во всех современных операционных системах. Audit позволяет отслеживать все события операционной системы семейства Linux, такие как:

– запуск и завершение работы системы;

– чтение/запись или изменение прав доступа к файлам;
– инициация сетевого соединения или изменение сетевых настроек;
– изменение информации о пользователе или группе;
– изменение даты и времени;
– запуск и остановка приложений;
– выполнение системных вызовов.

Подсистема Audit предоставляет подробную информацию о происходящем событии: дату и время возникновения события, пользователя, инициировавшего событие, тип события и его статус. На рисунке 4 в общем виде представлена схема работы подсистемы Audit. Отслеживание событий происходит с помощью триггеров, которые устанавливаются до и после всех функций, обрабатывающих системные вызовы. Когда происходит системный вызов, триггер срабатывает, подсистема аудита получает всю информацию о вызове и его параметрах и передаёт её демону Auditd. Далее управление передаётся функции, которая обрабатывает системный вызов. После завершения исполнения функции снова срабатывает триггер, и информация о событии поступает к подсистеме Audit и к демону Auditd.

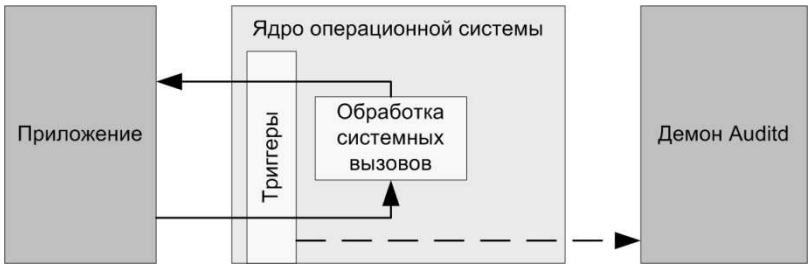


Рис. 4. Общая схема работы подсистемы Audit

Так как отслеживание каждого системного события приведёт к лишним временным затратам, триггеры по умолчанию отключены. Нужные триггеры активируются с помощью правил, которые позволяют задать название системного вызова, его состояние, пользователя и т.д. Управление триггерами осуществляется утилитой auditctl. С её помощью происходит создание, изменение и удаление триггеров.

Информацию, полученную от подсистемы Audit, демон Auditd записывает в журнальные файлы, которые находятся в каталоге /var/log/audit. Анализ этих файлов происходит с помощью специальных утилит, поставляемых вместе с самим демоном. Основной является утилита aureport, генерирующая отчёты из лог-файлов. С помощью дополнительных параметров утилиты можно получить интересующую в настоящий момент информацию, такую как: дата и время события, имя файла, номер системного вызова, статус, имя вызвавшего процесса и т.д.

Важной чертой системы Audit, которая используется в подсистеме событийного управления, является возможность централизованного сбора и хранения журнальных файлов. Этот функционал реализуется расширением audisp-remote.

На рисунке 5 представлена архитектура подсистемы событийного управления. Белые прямоугольники с толстой границей представляют собой типы узлов. Прямоугольники темно-серого цвета обозначают компоненты подсистемы событийного управления. Прямоугольники светло-серого цвета обозначают компоненты системы Slurm и подсистемы Audit. Прямоугольник со скруглёнными краями обозначает задание. Стрелками показана передача команд и данных. Пунктирными стрелками изображаются действия, которые выполняются из скриптов.

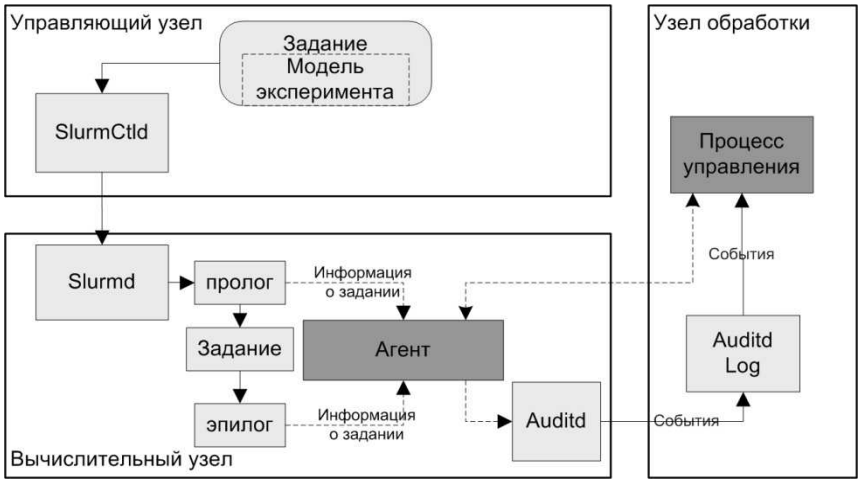


Рис. 5. Архитектура подсистемы событийного управления

В общем виде любое вычислительное задание выполняется следующим образом: сначала задание ставится в очередь системы планирования (демон SlurmCtld), затем происходит выделение ресурсов, далее управление передаётся демону Slurmd на выделенном вычислительном узле, после чего происходит запуск скрипта пролога, далее выполняется задание, потом исполняется скрипт эпилога, после этого выполнение вычислительного эксперимента завершено.

Подсистема событийного управления состоит из агента и процесса управления. Агент – это утилита, которая обеспечивает настройку окружения для отслеживания событий, происходящих в рамках вычислительного задания. Агент запускается на вычислительном узле и имеет связь с процессом управления. Процесс управления – это программа, обрабатывающая события от вычислительного узла и выполняющая реакции на эти события. На каждый вычислительный эксперимент создаётся свой процесс управления, поэтому разные эксперименты не влияют друг на друга.

Выше был описан запуск вычислительного задания без использования подсистемы событийного управления. Запуск вычислительного эксперимента с событийным управлением отличается тем, что при исполнении скриптов пролога и эпилога на вычислительном узле происходит вызов агента подсистемы событийного управления. Режим запуска с подсистемой событийного управления

вычислительного эксперимента выбирается при исполнении скрипта пролога, который проверяет наличии директории «.wbs» в рабочем каталоге вычислительного эксперимента. В данной директории находится модель вычислительного эксперимента. При вызове агента из скрипта пролога агент получает информацию о вычислительном задании (идентификатор задания, список вычислительных узлов, рабочая директория) и передаёт её процессу управления на узел обработки. На узле обработки создаётся экземпляр процесса управления, который будет обрабатывать события, поступающие от данного задания. Процесс управления считывает модель вычислительного эксперимента, по которой строит для каждого описанного события правила обнаружения и записывает результат в каталог «.wbs». Агент считывает файл с правилами и добавляет соответствующие триггеры в подсистему Audit. Далее происходит выполнение вычислительного задания. При срабатывании триггера «пойманное» событие отправляется на узел обработки. Обработка событий будет описана далее по тексту. После завершения вычислительного задания из скрипта эпилога происходит вызов агента, который считывает правила из файла и удаляет соответствующие триггеры. Вычислительный эксперимент завершён.

Отслеживание событий вычислительного процесса является малозатратной деятельностью и не потребляет значительного количества вычислительных ресурсов, в то время как реакция на события может быть длительным и ресурсоёмким процессом. Поэтому обработка данных в ходе реакции рассматривается как самостоятельный процесс, выполняемый параллельно основному на узле обработки.

Процесс управления получает события с вычислительного узла через демон Auditd Log. Полученные события отслеживаются на модели вычислительного эксперимента. При получении определённой комбинации событий происходит выполнение реакции. Чтобы реакции не влияли на процесс отслеживания событий, они выполняются в отдельном процессе.

**Проверка корректности построенной модели.** В процессе выполнения пользовательская программа генерирует события, которые отлавливаются с помощью подсистемы Audit. Собранную последовательность событий можно свернуть в более компактный вид и проверить на соответствие модели вычислительного эксперимента, которую построил пользователь. Это даёт возможность определить, выполняется ли вычислительный эксперимент так, как описал пользователь в модели или нет. Если нет, то указать место расхождения модели и реальности.

Пусть от некоторого процесса поступают события. Обозначим эти события символами $a_1$, $a_2$, $a_3$, $a_4$. Тогда в результате работы этого процесса может быть получена следующая последовательность событий $a_1a_2a_3a_2a_3a_2a_3a_4a_2a_2a_3a_2a_3a_4$. Данная последовательность событий может быть представлена в сокращённом виде (свёрнута) $a_1((a_2a_3)^3a_4)^2$. Для восстановления исходной последовательности из свёрнутого вида необходимо применить следующие правила. Актор $(...)^k$ обозначает, что последовательность событий, заключённая в скобки, должна быть повторена $k$ раз. Если $(...)^1$ или (…), тогда скобки должны быть опущены.

На рисунке 6а представлен алгоритм обработки списка. Существует исходный список, в котором записана необработанная последовательность. В цикле из начала исходного списка извлекается элемент и помещается в конец нового списка. К этому списку на каждой итерации цикла применяется операция свёртки.

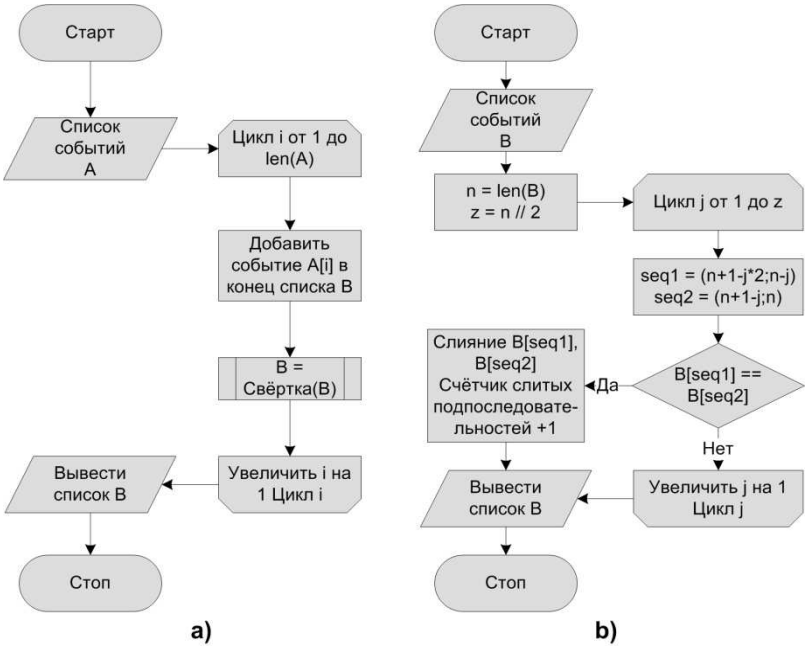


Рис. 6. Алгоритмы: a) Алгоритм обработки списка A; b) Алгоритм свёртки списка

На рисунке 6b представлен алгоритм свертки списка. Алгоритм свёртки заключается в поиске одинаковых подпоследовательностей в исходной последовательности и занесении их под скобку с увеличением счётчика на единицу. Для последовательности длиной $n$ максимально возможное количество сравнений подпоследовательностей $z = n \; div \; 2$ . Сравнения происходят с конца последовательности. В зависимости от номера сравнения вычисляются границы сравниваемых подпоследовательностей $(n+1-i*2, n-i)$ и $(n+1-i, n)$, где $i$ – счётчик итераций от 1 до $z$. После этого происходит сравнение подпоследовательностей и если они одинаковые, тогда подпоследовательности сливаются и счётчик слитых подпоследовательностей увеличивается на 1. Для каждой найденной подпоследовательности счётчик свой.

На рисунке 7 представлена работа алгоритма свёртки. В данном случае производятся 3 сравнения, начиная с конца последовательности:

1) сравниваются подпоследовательности ($a_3$) и ($a_4$). Они не равны;
2) сравниваются подпоследовательности ($a_4$, $a_2$) и ($a_3$, $a_4$). Они не равны;
3) сравниваются подпоследовательности ($a_2$, $a_3$, $a_4$) и ($a_2$, $a_3$, $a_4$). Они равны, поэтому подпоследовательности сливаются с увеличением счётчика слитых подпоследовательностей на 1.

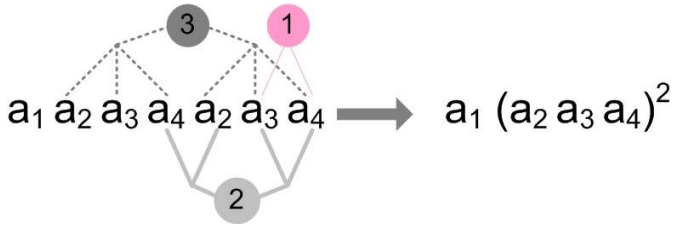


Рис. 7. Визуальное представление работы алгоритма свёртки

**Заключение.** В статье рассмотрен принцип работы событийного управления обработкой данных вычислительных экспериментов. На вычислительных кластерах используется различное программное обеспечение. Это могут быть как специализированные пакеты, так и программные коды, разработанные пользователями самостоятельно. Изменение исходных кодов с целью контроля событий представляет достаточно трудоёмкий процесс и не всегда выполнимый, так как программные пакеты могут поставляться в скомпилированном виде или может получиться так, что исходные коды недоступны или утеряны. Единственный способ обойти данные проблемы – это перехват событий. Перехват событий осуществляется с помощью подсистемы Audit. Альтернативным средством для отслеживания событий может выступать inotify, хотя inofity позволяет отслеживать только события файловой системы. Поэтому inotify не подходит для поставленной задачи. Главной особенностью разработанного решения является отсутствие необходимости в перепрограммировании существующих программных средств. Подсистема событийного управления обработкой данных вычислительных экспериментов реализована в рамках системы управления прохождением заданий WBS [5]. В дальнейшем авторами планируется разработка удобного пользовательского WEB-интерфейса, с помощью которого пользователи смогут реализовать возможности системы.

1. Джосан О.В. О визуализации научных данных для высокопроизводительных параллельных приложений // Тезисы конференции ПАВТ. – Н. Новгород, 2009.
2. Корж О.В. Автоматическое построение передаточных функций для систем визуализации распределенных данных на суперкомпьютерах // Параллельные вычислительные технологии (ПАВТ 2012): тр. междунар. науч. конф. (Новосибирск, 26–30 марта 2012). – Челябинск, изд. центр ЮУрГУ 2012. С. 726–726.
3. Leontyev D.V., Kharitonov D.I., Tarasov G.V The data processing model of computational experiments // 20th Conference Scientific Services and Internet: CEUR-WS, 2018. Vol. 2260. P. 373–386.
4. Leontyev D.V., Kharitonov D.I., Tarasov G.V. Imperative programs behavior simulation in terms of compositional Petri nets // International Journal of Computer Networks & Communications (IJCNC). 2018. Vol. 10, №1.
5. Одякова Д.С., Тарасов Г.В., Харитонов Д.И. Система WBS как расширенный инструмент управления вычислительной средой // Суперкомпьютерный форум «Суперкомпьютерные технологии в образовании, науке и промышленности» 26–28 ноября 2012 г., Нижний Новгород). Нижегородский государственный университет им. Н.И. Лобачевского, 2012.

**Транслитерация**

1. Dzhosan O.V. O vizualizacii nauchnyh dannyh dlya vysokoproizvoditel'nyh parallel'nyh prilozhenij // Tezisy konferencii PAVT. – N. Novgorod, 2009.
2. Korzh O.V. Avtomaticheskoe postroenie peredatochnyh funkcij dlya sistem vizualizacii raspredelennyh dannyh na superkomp'yuterah // Parallel'nye vychislitel'nye tekhnologii (PAVT 2012): tr. mezhdunar. nauch. konf. (Novosibirsk, 26–30 marta 2012). – Chelyabinsk, izd. centr YUUrGU 2012. P. 726–726.
3. Odyakova D.S., Tarasov G.V., Haritonov D.I. Sistema WBS kak rasshirennyj instrument upravleniya vychislitel'noj sredoj // Superkomp'yuternyj forum «Superkomp'yuternye tekhnologii v obrazovanii, nauke i promyshlennosti» 26–28 noyabrya 2012 g., Nizhnij Novgorod). Nizhegorodskij gosudarstvennyj universitet im. N.I.Lobachevskogo, 2012.



**Для цитирования:** Леонтьев Д.В., Харитонов Д.И., Одякова Д.С., Парахин Р.В. Автоматизация постобработки данных вычислительных экспериментов // Территория новых возможностей. Вестник Владивостокского государственного университета экономики и сервиса. 2019. Т. 11, № 4. С. 207–218.

**For citation:** Leontiev D.V., Kharitonov D.I., Odyakova D.S., Parakhin R.V. Automating computational experiments data post-processing, *The Territory of New Opportunities. The Herald of Vladivostok State University of Economics and Service*, 2019, Vol. 11, № 4, pp. 207–218.

DOI dx.doi.org/10.24866/VVSU/2073-3984/2019-4/207-218

Дата поступления: 11.11.2019.