ВЕСТНИК
ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
МОРСКОГО И РЕЧНОГО ФЛОТА ИМЕНИ АДМИРАЛА С. О. МАКАРОВА

DOI: 10.21821/2309-5180-2017-9-4-892-901

DISTRIBUTED COMPUTING FOR OPTIMIZATION OF OPERATIONAL MANAGEMENT OF THE TRANSPORT CONVEYOR OF LOCK SYSTEM

V. A. Kuznetsov

Admiral Makarov State University of Maritime and Inland Shipping,
St. Petersburg, Russian Federation

On the inland waterways of Russia, which are part of a single transport system, there are more than 150 locks, most of which are integrated in the lock systems. The automation measures and, inevitably, the optimization of operational management of the transport conveyor of lock system, are able, first, reduce operation of lock canals and transportation costs, by saving energy and time resources, second, in the case of a high ships flow, increase the capacity of the canals to relieve them. At the moment systems implementing automated dispatching passing of the whole lock canal, not created. It is concluded that this requires the use of the system that allows the automatic exchange of information between ships and shore services, as well as high-performance multicomputer systems, features of creation of which are considered in the article.

As the environment of implementing is proposed distributed computer system that allows you to combine multiple computers using a universal program shell of parallelizing for cooperative processing. Its model and implementation feature is presented in the article. It is noted that its main purpose is organization of efficient execution of user application solution, using for them compute capacity of the computer devices that allow parallelization. Their integration into a single compute cluster, when solving the problems of optimization of operational management of lock system, gives the following advantages. First, allows to form hardware-software complexes of any computing power, the required value of which is determined by the parameters of the lock canal, and, if necessary, to increase computing power during work; second, it increases system fault tolerance, which is an important factor in the operation of such objects.

Keywords: optimization of operational management, lock canal, distributed computing system, parallel algorithms, cluster.

For citation:
Kuznetsov, Vitaliy A. "Distributed computing for optimization of operational management of the transport conveyor of lock system." *Vestnik Gosudarstvennogo universiteta morskogo i rechnogo flota imeni admirala S. O. Makarova* 9.4 (2017): 892–901. DOI: 10.21821/2309-5180-2017-9-4-892-901.

УДК 004.42

РАСПРЕДЕЛЁННЫЕ ВЫЧИСЛЕНИЯ ДЛЯ ОПТИМИЗАЦИИ ОПЕРАТИВНОГО УПРАВЛЕНИЯ ТРАНСПОРТНЫМ КОНВЕЙЕРОМ ШЛЮЗОВАННОЙ СИСТЕМЫ

В. А. Кузнецов

ФГБОУ ВО «ГУМРФ имени адмирала С. О. Макарова»,
Санкт-Петербург, Российская Федерация

На внутренних водных путях России, являющихся частью единой транспортной системы, насчитывается более 150 шлюзов, большинство из которых интегрированы в шлюзованные системы. Мероприятия по автоматизации и, как следствие, оптимизации оперативного управления судопропуском способны, во-первых, сократить затраты на эксплуатацию шлюзованных судоходных каналов и себестоимость перевозок за счёт экономии энергетических и временных ресурсов, во-вторых, в случае повышенного судопотока, увеличить пропускную способность каналов для их разгрузки. На данный момент систем, реализующих автоматизированную диспетчеризацию шлюзованного судоходного канала в целом, не создано. Делается вывод, о том, что для этого необходимо использовать системы, обеспечивающие автоматический обмен информацией между судами и береговыми службами, а также высокопроизводительные многомашинные вычислительные комплексы, особенности построения которых рассматриваются в статье.

В качестве среды реализации предлагается распределённая вычислительная система, которая позволяет объединять множество компьютеров, использующих универсальную программную оболочку распараллеливания для совместной обработки данных. Описывается её модель и особенности функционирования. Отмечается, что основным назначением программной оболочки является организация эффективного выполнения прикладного решения пользователя с использованием вычислительной мощности устройств компьютера, допускающих распараллеливание. Их объединение в единый вычислительный кластер при решении вопросов оптимизации оперативного управления шлюзованной системы даёт следующие преимущества: во-первых, позволяет формировать аппаратно-программные комплексы любой вычислительной мощности, требуемый уровень которой определяется параметрами судоходного канала, и при необходимости наращивать её в процессе эксплуатации; во-вторых, повышает отказоустойчивость системы, что является важным фактором при функционировании такого рода объектов.

Ключевые слова: оптимизация оперативного управления, шлюзованные судоходные каналы, распределенные вычислительные системы, параллельные алгоритмы, кластер.

Для цитирования:
Кузнецов В. А. Распределённые вычисления для оптимизации оперативного управления транспортным конвейером шлюзованной системы / В. А. Кузнецов // Вестник Государственного университета морского и речного флота имени адмирала С. О. Макарова. — 2017. — Т. 9. — № 4. — С. 892–901. DOI: 10.21821/2309-5180-2017-9-4-892-901.

Введение

В статье [1] рассматриваются вопросы оптимизации оперативного управления транспортным конвейером шлюзованного судоходного канала (ШСК). Обосновывается возможность реализации автоматизированной диспетчеризации судопропуска на основе взаимодействия имитационной модели и алгоритма управления в реальном масштабе времени, конечной целью которых является расписание проводки судов. До настоящего времени подобных систем, работающих в рамках ШСК в целом, не создано. Делается вывод о том, что для этого необходимо использовать системы, обеспечивающие автоматический обмен информацией между судами и береговыми службами, а также параллельные алгоритмы обработки данных. В качестве среды реализации предложена универсальная программная оболочка распараллеливания (ПОР). Рассматриваются её модель и особенности функционирования. Основным назначением ПОР является эффективное исполнение прикладного решения пользователя. Для этих целей программная оболочка использует весь потенциал подключенных к компьютеру вычислительных устройств, допускающих распараллеливание.

Описанный подход позволяет сформировать оптимальное расписание по выбранному критерию, например, минимальный расход энергоресурсов или минимум совокупных потерь провозной способности флота при прохождении канала, что, в свою очередь, позволит снизить затраты на эксплуатацию шлюзованной системы и (или) себестоимость перевозок. В процессе функционирования автоматизированной системы оптимизации оперативного управления судопропуском критерий можно динамически изменять, тем самым адаптируясь к ситуации на канале. Так, при повышенном судопотоке для разгрузки ШСК целесообразно направить оптимизацию расписания проводки судов на увеличение его пропускной способности.

Несомненно, использование ПОР позволяет существенно повысить эффективность вычислений, однако в связи с физическими ограничениями полностью решить вопросы производительности, используя ресурсы одного компьютера, не представляется возможным [2]. Решить указанную проблему можно за счёт объединения компьютеров в единую распределённую вычислительную систему (РВС) или кластер [3]. Следует отметить, что реализация параллельной обработки данных на устройствах одного компьютера и в рамках РВС — концептуально разные задачи [4].

Особенности построения РВС

Вычислительная мощность кластера зависит от количества узлов в нём, их аппаратных и программных ресурсов, а также алгоритма распределения (балансировки) нагрузки между ними. На практике организация эффективной РВС — достаточно трудоёмкая задача. Это объясняется

наличием множества факторов, часто взаимосвязанных между собой, которые влияют на её производительность. Ограничения служб и алгоритмов, связанные с использованием определённого типа оборудования, технологий или ПО, зачастую выступают своего рода «тормозом» увеличения количества узлов [5]. Степень проблемы зависит от требований конкретной системы. Так, для открытых глобальных РВС характерно разнообразие технических и программных средств, входящих в неё компонентов, что требует применения кроссплатформенных решений для обеспечения совместной работы приложений на широком диапазоне платформ [2], [6]. Как правило, в локальных вычислительных кластерах используется однородное оборудование и ПО, подобранное для максимально эффективного решения поставленных задач.

От количества узлов в РВС зависит нагрузка на сервер, что накладывает ограничения на максимальное количество узлов, поддерживаемое кластером [7]. Дальнейшее увеличение их числа возможно за счёт повышения пропускной способности сетей и мощности сервера. Использование децентрализованной архитектуры, в которой узлы взаимодействуют друг с другом напрямую, без участия единого центра, позволяет полностью решить вопросы нехватки производительности серверов. При этом, в силу сложной организации такой РВС по сравнению с централизованной, возрастают требования к узлам системы и нагрузка на сети передачи данных.

Сбои в работе вычислительных систем, построенных на базе единичных компьютеров, зачастую приводят к полному отказу в работоспособности. Кластеры более устойчивы к подобным отказам за счёт избыточности аппаратных и программных ресурсов. При выходе из строя одного из узлов выполняемый им объём работ распределяется между другими компонентами системы, что обеспечивает её бесперебойную работу. Функционирование РВС с централизованной архитектурой зависит от сервера, выход из строя которого приводит к остановке работы всей системы. В такой ситуации для повышения отказоустойчивости необходимо использовать кластеры серверов, обеспечивающих единый функционал.

Наряду с характеристиками узлов и их количеством в кластере одним из важнейших факторов, влияющих на производительность и скорость работы РВС, является алгоритм распределения нагрузки между узлами [8], [9]. Стратегия балансировки должна учитывать множество факторов, основными из которых являются: размерность и сложность задач, различия аппаратных и программных ресурсов узлов системы, пропускная способность средств коммуникации [7], [10]. Также распределение необходимо выполнять таким образом, чтобы обеспечить оптимальное количество обращений между компонентами системы, это позволит уменьшить временные затраты на синхронизацию, снизить нагрузку на сети передачи данных и сервер.

Балансировка бывает *статической* и *динамической* [9]. Статическая балансировка выполняется за счёт априорного анализа конфигурации РВС, её структуры и характеристик, входящих в неё компонентов. Основной недостаток такого подхода заключается в том, что нагрузка не меняется в ходе работы системы, при этом не учитываются такие показатели, как текущая загруженность узлов и сетей передачи данных, сбои в работе её компонентов, изменение количества узлов или их конфигурации. При динамической балансировке нагрузка на узлы рассчитывается в процессе функционирования вычислительного кластера с учётом его текущего состояния, что, в свою очередь, предполагает реализацию системы мониторинга. Динамическая балансировка также позволяет использовать программное обеспечение, работоспособность которого не зависит от структуры распределённой системы [7].

Построение РВС на базе ПОР

Для организации совместных распределённых вычислений двух и более ПОР реализуется РВС централизованной архитектуры, что означает наличие единого центра управление (ЦУ) вычислительным процессом. ЦУ обеспечивает подключение и взаимодействие с множеством ПОР через сеть, а также максимально эффективное распределение вычислительной нагрузки между ними.

Изменения в реализации ПОР касаются переноса функций по контролю и организации процесса решения пользовательской задачи (ПЗ) от программной оболочки к ЦУ и добавления модуля

для работы с ним. При этом обработка взаимодействия с аппаратными ресурсами компьютера и пользователя не изменилась. Это означает, что приём и формирование ПЗ, а также вывод результата его решения, по-прежнему выполняет ПОР. После инициализации ПЗ отправляется ЦУ. Решение задачи в ЦУ осуществляется за счёт её декомпозиции на части, именуемые *сегментами*, каждый из которых передаётся на решение одному из свободных узлов кластера. Создание сегмента пользовательской задачи (СПЗ) осуществляется с помощью dll-библиотеки, которая реализуется для каждого типа задачи. При выборе оптимального размера СПЗ анализируются характеристики, информация об оборудовании и вычислительной мощности узла, для которого этот сегмент формируется. Как только СПЗ будет решён, ПОР отправляет результат ЦУ, который обрабатывает его и в случае получения конечного решения ПЗ возвращает результат отправителю этой задачи. Таким образом, каждый участник РВС не только предоставляет вычислительную мощность своего компьютера, но также может сам воспользоваться ресурсами системы.

Рассмотрим подробнее изменения в работы ПОР, в качестве узла РВС, и модель функционирования вычислительного кластера.

Алгоритм работы ПОР совместно с ЦУ

ПОР представляет собой совокупность взаимодействующих модулей [1]. Связующим звеном этих модулей является ядро программной оболочки (ЯПО). Вся основная логика работы ПОР заложена во взаимодействии ЯПО с модулями устройств (МУ), которые выполняют основную работу вычислений ПОР — решение ПЗ. Алгоритм решения реализуется в виде dll-библиотек, разработанных для каждого типа устройства, работающего с ПОР. Каждая библиотека должна содержать значения параметров, требуемых для настройки среды программной оболочки, таких как размер массива бинарных данных задачи и результата её решения; идентификатор задачи и др. Также необходимо реализовать ряд экспортных функций, обеспечивающих выполнение следующих операций:

– Score — оценка производительности устройства;

– Parser — проверка, обработка и формирование данных ПЗ;

– Cut — формирование данных подзадачи;

– SimSim — решение подзадачи;

– Result и ResultBuild — формирование общего решения ПЗ.

Опишем работу ЦУ и ПОР в виде пошагового алгоритма. Следует отметить, что СПЗ — это часть ПЗ, которую отправляет ЦУ для решения ПОР, в свою очередь, подзадача — часть СПЗ, которая отправляется МУ для обработки.

Этап 1. Создание МУ, отвечающих за взаимодействие с конкретным аппаратным средством на основе информации, сформированной в результате предварительного анализа вычислительных устройств компьютера, с которыми поддерживает работу ПОР.

Этап 2. Подключение ПОР к ЦУ, при этом передаётся конфигурация узла.

Этап 3. Инициализация задачи ПОР посредством считывания и обработки данных конфигурационного файла, созданного пользователем, который содержит информацию о типе задачи и бинарные данные задачи, соответствующие выбранному типу. Передача сформированной задачи ЦУ.

Этап 4. ЦУ отправляет задачу на решение множеству подключённых к ЦУ ПОР, при этом для каждого узла на основании его конфигурации формируются данные СПЗ.

Этап 5. Решение полученного сегмента задачи в ПОР:

– *шаг* 1 — с помощью функции Score, вызванной из соответствующей dll-библиотеки, оценивается скорость решения текущего сегмента задачи на определенном устройстве;

– *шаг* 2 — ЯПО отправляет задачу на решение множеству параллельно работающих МУ, при этом, в зависимости от оценки производительности устройства, модуль Cute формирует данные подзадачи для каждого МУ;

– *шаг* 3 — в процессе решения подзадачи модуль SimSim осуществляет дополнительное распараллеливание вычислительного процесса, используя особенности аппаратного средства;

– *шаг* 4 — после решения подзадачи МУ отправляет результат ЯПО;

– *шаг* 5 — формирование результата решения сегмента задачи на основе информации, полученной от МУ, и отправка его ЦУ.

Этап 6. Формирование общего результата решения ПЗ, на основе информации, полученной от узлов кластера, и отправка его в ПОР, который её инициализировал.

Модель функционирования РВС на базе ПОР

При формировании требований к параллельным и распределённым алгоритмам наиболее важным является определение последовательности определённых событий для каждого вычисления программы. Это позволяет для описания модели организации совместных распределённых вычислений двух и более ПОР использовать язык темпоральной логики линейного времени (LTL), в котором явно учтен феномен времени. Каждое событие будет характеризоваться булевой переменной, которая принимает значение «Истина» тогда и только тогда, когда наступает событие.

В LTL, помимо булевых логических связок, для описания причинно-следственной зависимости событий во времени используются темпоральные операторы:

– Xφ «в следующий момент времени», указывает на то, что φ выполняется на следующем состоянии пути;

– Fφ «когда-то в будущем», указывает на то, что φ будет соблюдаться в каком-то последующем состоянии пути;

– Gφ «всегда», указывает на то, что φ выполняется в каждом состоянии пути;

– φUψ «до тех пор, пока», указывает на то, что φ выполняется до тех пор, пока не станет верно ψ;

– φRψ «высвободить, открепить», указывает на то, что ψ может перестать быть верным только после того, как станет верно φ.

Также введём следующие атомарные высказывания, соответствующие основным событиям вычислений программы:

– try_task_i — *i*-й пользователь собирается отправить задачу;
– add_task_i — *i*-я ПЗ добавляется в очередь задач;
– del_task_i — *i*-я ПЗ удаляется из очереди задач;
– $send\_task_i$ — *i*-я ПЗ отправляется на решение ЯПО;
– $create\_segment_i$ — *i*-я СПЗ формируется из ПЗ;
– $send\_segment_i$ — *i*-я СПЗ отправляется на решение МУ;
– $create\_subtask_i$ — *i*-я подзадача формируется из СПЗ;
– $solve\_subtask_i$ — *i*-й МУ решил подзадачу;
– $recv\_subtask_i$ — *i*-й результат подзадачи МУ отправляется ЯПО;
– $res\_segment_i$ — формируется *i*-й результат СПЗ;
– $recv\_segment_i$ — результат *i*-й СПЗ отправляется в ЦУ;
– res_task_i — формируется *i*-й результат ПЗ;
– end_segment — получен конечный результат СПЗ;
– end_task — получен конечный результат ПЗ;
– free_module — МУ свободен;
– busy_module — МУ занят;
– free_core — ЯПО свободен;
– busy_core — ЯПО занят;
– empty — очередь задач пуста.

Рассмотрим математическую модель программной оболочки, построенную на основе языка LTL.

1. Всякий раз, когда пользователь собирается запустить ПЗ, ЦУ добавляет себе ПЗ в очередь задач

$$G\left(\text{try\_task}_i \rightarrow \text{add\_task}_j\right). \quad (1)$$

2. Пока очередь задач не окажется пустой, ЦУ будет отправлять задачи на решение свободным ЯПО:

$$G\left(\left(\text{free\_core} \rightarrow \text{send\_task}_i\right) \ U \ \text{empty}\right). \quad (2)$$

3. Всякий раз, когда ЦУ отправляет задачу ЯПО, формируется СПЗ для этого ЯПО:

$$G\left(\text{send\_task}_i \rightarrow \text{X create\_segment}_j\right). \quad (3)$$

4. Всякий раз, когда для ЯПО формируется СПЗ, он становится занятым:

$$G\left(\text{create\_segment}_\text{i} \rightarrow \text{busy\_core}\right). \quad (4)$$

5. Всякий раз, когда ЯПО становится занятым, когда-нибудь должен быть найден конечный результат решения СПЗ:

$$G\left(\text{busy\_core} \rightarrow F \ \text{end\_segment}\right). \quad (5)$$

6. Пока не получен конечный результат решения СПЗ, ЯПО будет отправлять СПЗ на решение свободным МУ:

$$G\left(\left(\text{free\_module} \rightarrow \text{send\_segment}_\text{i}\right) \ \rightarrow \ \text{! end\_segment}\right). \quad (6)$$

7. Всякий раз, когда ЯПО отправляет СПЗ на решение МУ, формируется подзадача для этого МУ:

$$G\left(\text{send\_segment}_\text{i} \rightarrow X \ \text{create\_subtask}_j\right). \quad (7)$$

8. Всякий раз, когда для МУ формируется подзадача, он становится занятым:

$$G\left(\text{create\_subtask}_\text{i} \rightarrow \text{busy\_module}\right). \quad (8)$$

9. Всякий раз, когда МУ становится занятым, должна когда-нибудь решиться подзадача:

$$G\left(\text{busy\_module} \rightarrow \text{F solve\_subtask}_\text{i}\right). \quad (9)$$

10. После решения подзадачи МУ возвращает результат ЯПО и становиться свободным:

$$G\left(\text{solve\_subtask}_i \rightarrow X \left(\text{recv\_subtask}_j \wedge \text{free\_module}\right)\right). \quad (10)$$

11. Всякий раз, когда ЯПО получает результат решения подзадачи от МУ, ЯПО формирует результат соответствующей СПЗ:

$$G\left(\text{recv\_subtask}_i \rightarrow X \ \text{res\_segment}_j\right). \quad (11)$$

12. Всякий раз, когда ЯПО формирует результат решения СПЗ, оболочка проверяет, получен или нет её конечный результат. Если этот результат получен, то ЯПО возвращает результат СПЗ ЦУ и становится свободным:

$$G\left(\left(\text{res\_segment}_i \rightarrow \text{end\_segment}\right) \rightarrow X\left(\text{recv\_segment}_j \rightarrow \text{free\_core}\right)\right). \quad (12)$$

13. Всякий раз, когда ЦУ получает результат решения СПЗ от ЯПО, ЦУ формирует результат соответствующей ПЗ:

$$G\left(\text{recv\_segment}_i \rightarrow X \ \text{res\_task}_j\right). \quad (13)$$

14. Всякий раз, когда ЦУ формирует результат решения ПЗ, выполняется проверка того, получен или нет конечный результат. Если этот результат получен, то ПЗ удаляется из очереди:

$$G\left(\left(\text{res\_task}_i \wedge \text{end\_task}\right) \wedge \text{del\_task}_j\right). \quad (14)$$

Полученные зависимости темпоральной логики позволяют представить псевдокод основных процессов модели (листинги 1 – 3). Для каждой операции псевдокода ставится счетчик. Следует заметить, что псевдокод для ЯПО и МУ являются инвариантными по отношению ко всем компьютерам и вычислительным устройствам соответственно:

```
        while (true) {
1.1.        if (try_task)
1.2.            add_task();
1.3.        if (!empty && free_core)
1.4.            send_task();
1.5.        if (recv_segment()) {
1.6.            res_task();
1.7.            if (end_taks)
1.8.                del_task();
            }
        }
```

Листинг 1. Псевдокод ЦУ

В соответствии с построенным псевдокодом строится размеченная система переходов LTS (Labelled Transition System), где учитываются основные состояния процессов модели:

– счетчики выполнения команд;

– состояние решения подзадачи (решено / не решено);

– состояние нахождения конечного решение СПЗ (найдено / не найдено);

– состояние ЯПО (занят / свободен);

– состояние модуля устройства (занят / свободен).

На рис. 1 изображены схемы LTS каждого отдельного процесса: единственного — для ЦУ и множества процессов ЯПО и МУ. В узлах через запятую показаны значения основных состояний процессов модели.

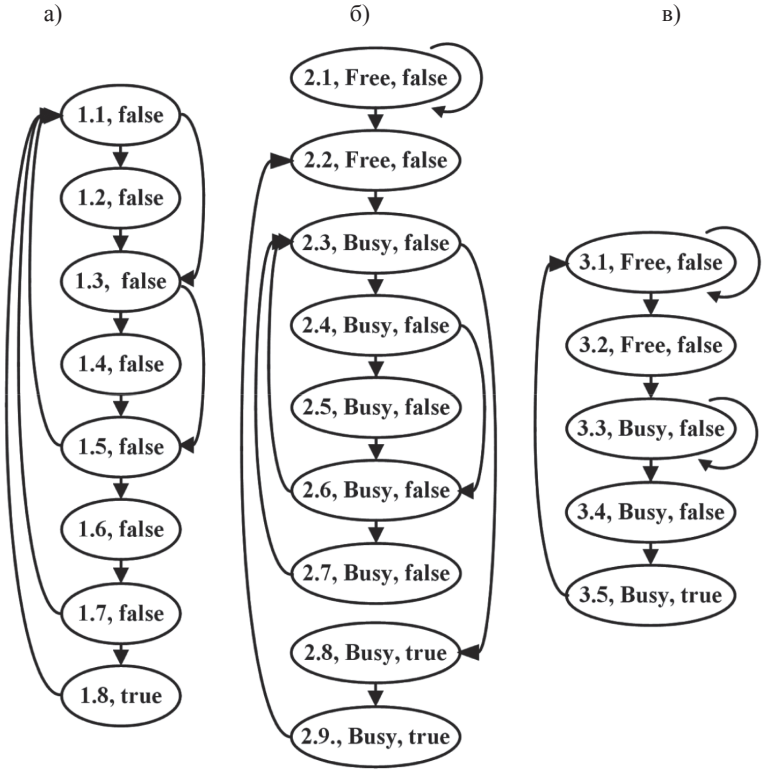

Рис. 1. LTS основных процессов модели: *а* — ЦУ; *б* — ЯПО; *в* — МУ

```
        while (true) {
2.1.        if (create_segment()) {
2.2.            status = busy_core;
2.3.            while (!end_segment) {
2.4.                if (free_module)
2.5.                    send_segment();
2.6.                if (recv_subtask())
2.7.                    res_segment();
                }
2.8.            recv_segment();
2.9.            status = free_core;
            }
        }
```

Листинг 2. Псевдокод ЯПО

```
        while (true) {
3.1.        if (create_subtask()) {
3.2.            status = busy_module;
3.3.            while (!solve_subtask) {}
3.4.            recv_subtask();
3.5.            status = free_module;
            }
        }
```

Листинг 3. Псевдокод МУ

Поскольку программная реализация должна обеспечивать работу со многими процессами МУ и ЯПО, рассмотрим LTS-схемы демонстрирующие динамику взаимодействия двух процессов МУ и одного процесса ЯПО (рис. 2), а также двух процессов ЯПО и одного процесса ЦУ (рис. 3).

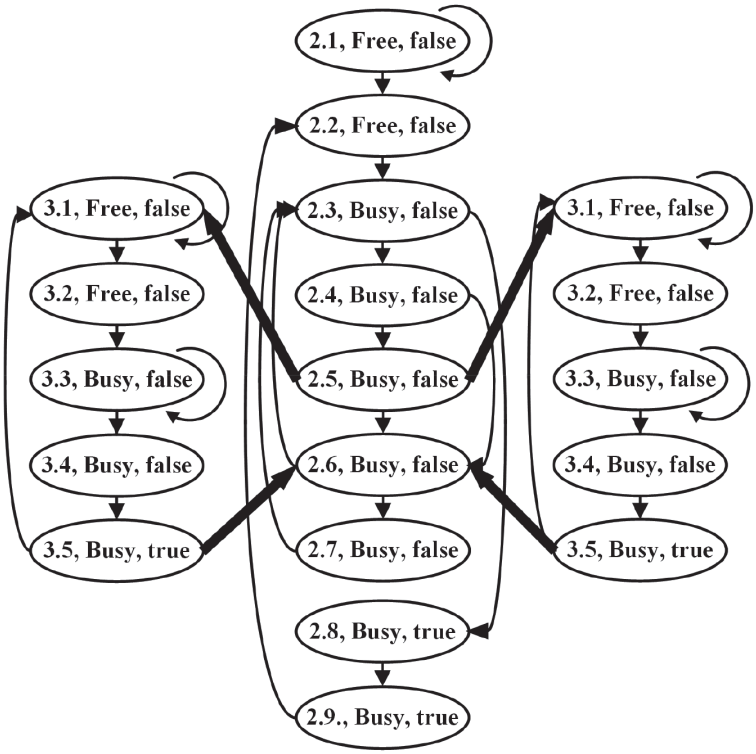

Рис. 2. LTS взаимодействия процесса ЯПО с двумя процессами МУ

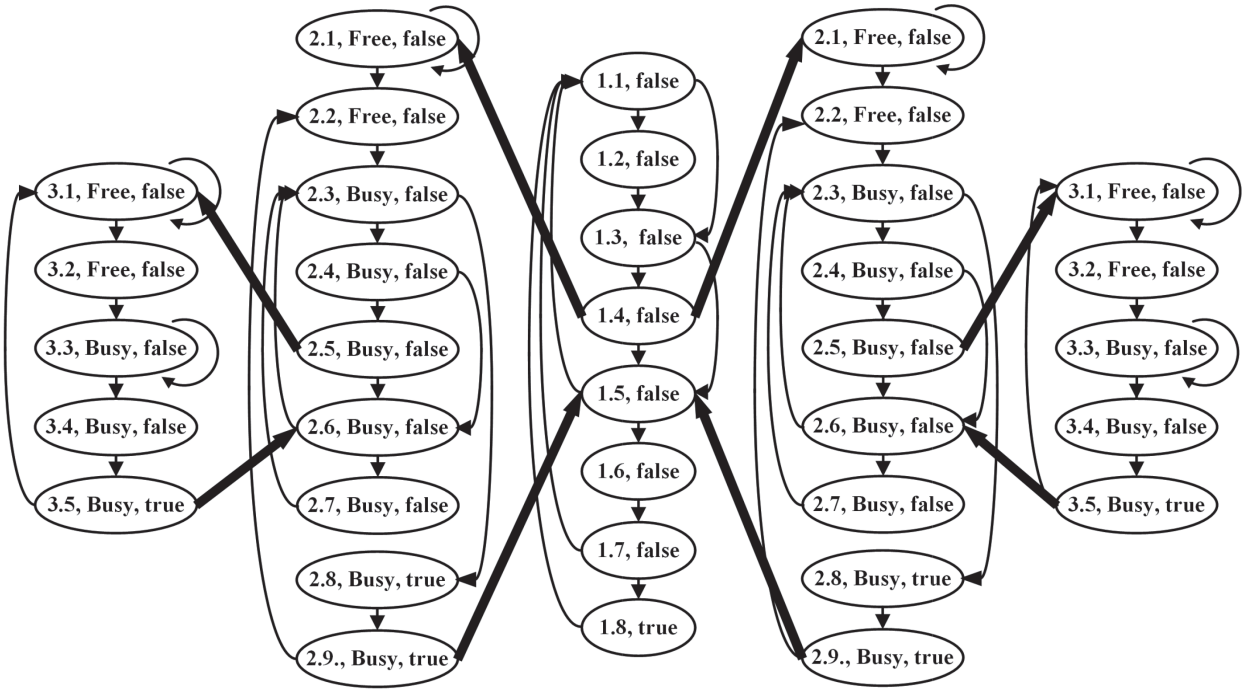

Рис. 3. LTS взаимодействия процесса ЦУ с двумя процессами ЯПО, работающих с одним процессом МУ

ЯПО управляет работой процессов МУ, выполняемых параллельно. Взаимодействие МУ и ЯПО, осуществляемое во временные отметки отправки (блок состояния ЯПО — «2.5, Busy, false») и получения (блок состояния ЯПО — «2.6, Busty, false») результата решения подзадачи МУ, должно быть синхронизировано. На схеме синхронизация процессов показана жирными линиями.

ЦУ аналогичным образом управляет работой процессов ЯПО, выполняемых параллельно. Взаимодействие ЯПО и ЦУ, осуществляемое во временные отметки отправки (блок состояния ЦУ — «1.4, false») и получения (блок состояния ЦУ — «1.5, false») результата решения подзадачи СПЗ, также должно быть синхронизировано.

Выводы

Использование технологий распределённой обработки данных для повышения производительности ПОР за счёт их объединения в единый вычислительный кластер при автоматизированной диспетчеризации судопропуска ШСК даёт следующие преимущества. Во-первых, позволяет реализовывать аппаратно-программный комплекс любой вычислительной мощности, требуемый уровень которой определяется параметрами судоходного канала, и при необходимости наращивать её в процессе эксплуатации, во-вторых, повышает отказоустойчивость системы, что является важным фактором при функционировании такого рода объектов.

СПИСОК ЛИТЕРАТУРЫ

1. *Егоров А. Н.* Распараллеливание вычислений для оптимизации оперативного управления транспортным конвейером шлюзованной системы / А. Н. Егоров, В. А. Кузнецов // Вестник Государственного университета морского и речного флота имени адмирала С. О. Макарова. — 2016. — № 2 (36). — С. 214–224. DOI: 10.21821/2309-5180-2016-8-2-214-224.

2. *Алпатов А. Н.* Развитие распределенных технологий и систем / А. Н. Алпатов // Перспективы Науки и Образования. — 2015. — № 2 (14). — С. 60–66.

3. In search of clusters. — Second edition. — Upper Saddle River: Prentice Hall, 1998. — 578 p.

4. *Камерон Х.* Параллельное и распределенное программирование с использованием C++ / Х. Камерон, Х. Трейси. — М: Вильямс, 2004 — 672 с.

5. *Сухорослов О. В.* Организация вычислений в гетерогенных распределенных средах / О. В. Сухорослов // Известия ЮФУ. Технические науки. — 2016. — № 12 (185). — С. 115–130. DOI: 10.18522/2311-3103-2016-12-115130.

6. *Truonga H.-L.* Towards the Realization of Multi-dimensional Elasticity for Distributed Cloud Systems / H.-L. Truonga, S. Dustdara, F. Leymann // Procedia Computer Science. — 2016. — Vol. 97. — Pp. 14–23. DOI: 10.1016/j.procs.2016.08.276.

7. *Цветков В. Я.* Проблемы распределенных систем / В. Я. Цветков, А. Н. Алпатов // Перспективы Науки и Образования. — 2014. — № 6 (12). — С. 31–36.

8. *Metaweia M. A.* Load balancing in distributed multi-agent computing systems / M. A. Metaweia, S. A. Ghoneim, S. M. Haggag, S. M. Nassar // Ain Shams Engineering Journal. — 2012. — Vol. 3. — Is. 3. — Pp. 237–249. DOI: 10.1016/j.asej.2012.03.001.

9. *Daryapurkar A.* Efficient Load Balancing Algorithm in Cloud Environment / A. Daryapurkar, M. V. M. Deshmukh // International Journal Of Computer Science And Applications. — 2013. — Vol. 6. — No. 2. — Pp. 308–312.

10. *Олзоева С. И.* Оптимальная балансировка нагрузки в GRID-системах / С. И. Олзоева, В. Ц. Мархоев, А. Г. Олзоева // Вестник Бурятского государственного университета. — 2012. —№ SB. — С. 220–223.

REFERENCES

1. Yegorov, Alexander Nikolayevich, and Vitaliy Aleksandrovich Kuznetsov. “Parallel computing for optimization of operational management of the transport conveyor of lock system.” *Vestnik Gosudarstvennogo universiteta morskogo i rechnogo flota imeni admirala S.O. Makarova* 2(36) (2016): 214–224. DOI: 10.21821/2309-5180-2016-8-2-214-224.

2. Alpatov, Aleksei Nikolaevich. “The development of distributed technologies and systems.” *Perspectives of Science and Education* 2(14) (2015): 60–66.

3. *In search of clusters.* Second edition. Upper Saddle River: Prentice Hall, 1998.

4. Kameron, Kh., and Kh. Treisi. *Parallel’noe i raspredelennoe programmirovanie s ispol’zovaniem C++.* M: Vil’yams, 2004.

5. Sukhoroslov, Oleg Viktorovich. “Organization of computations in heterogenous distributed environments.” *Izvestiya SFedU. Engineering Sciences* 12(185) (2016): 115–130. DOI: 10.18522/2311-3103-2016-12-115130.

6. Truonga, Hong-Linh, Schahram Dustdar, and Frank Leymann. “Towards the Realization of Multi-dimensional Elasticity for Distributed Cloud Systems”. *Procedia Computer Science* 97 (2016): 14–23. DOI: 10.1016/j.procs.2016.08.276.

7. Tsvetkov, Viktor Yakovlevich, and Aleksei Nikolaevich Alpatov. “Problems of distributed systems.” *Perspectives of Science and Education* 6(12) (2014): 31–36.

8. Metaweia, Maha A., Salma A. Ghoneima, Sahar M. Haggaga, and Salwa M. Nassar. “Load balancing in distributed multi-agent computing systems.” *Ain Shams Engineering Journal* 3.3 (2012): 237–249. DOI: 10.1016/j.asej.2012.03.001.

9. Daryapurkar, A., A. Daryapurkar, and M. V. M. Deshmukh. “Efficient Load Balancing Algorithm in Cloud Environment.” *International Journal Of Computer Science And Applications* 6.2 (2013): 308–312.

10. Olzoeva, Seseg Ivanovna, Vitaliy Tsydenovich Markhoev, and Ayuna Gennadevna Olzoeva. “The optimal load balancing in GRID systems.” *Vestnik Buryatskogo gosudarstvennogo universiteta* SB (2012): 220–223.

ИНФОРМАЦИЯ ОБ АВТОРЕ

Кузнецов Виталий Александрович — аспирант
Научный руководитель:
Егоров Александр Николаевич.
ФГБОУ ВО «ГУМРФ имени
адмирала С. О. Макарова»
198035, Российская Федерация, Санкт-Петербург,
ул. Двинская, 5/7
e-mail: *kuznecov.v.spb@mail.ru, kaf_vsi@gumrf.ru*

INFORMATION ABOUT THE AUTHOR

Kuznetsov, Vitaliy A. — Postgraduate.
Supervisor:
Yegorov, Alexander N.
Admiral Makarov State University of Maritime
and Inland Shipping
5/7 Dvinskaya Str., St. Petersburg, 198035,
Russian Federation
e-mail: *kuznecov.v.spb@mail.ru, kaf_vsi@gumrf.ru*

Статья поступила в редакцию 6 июля 2017 г.
Received: July 6, 2017.