Omega: планировщики задач вычислительных кластеров

Лев Хорошанский

Планирование

Часть современного мира, касающаяся компьютеров и вычислений на них, так или иначе связана с решением конкретных задач, для чего требуются ресурсы и их грамотное использование. Метод, согласно которому вычислительным машинам распределяются решаемые задачи, называется **планированием выполнения задач**, тогда как алгоритм и/или процесс, занимающийся планированием задач, называется **планировщиком задач**.

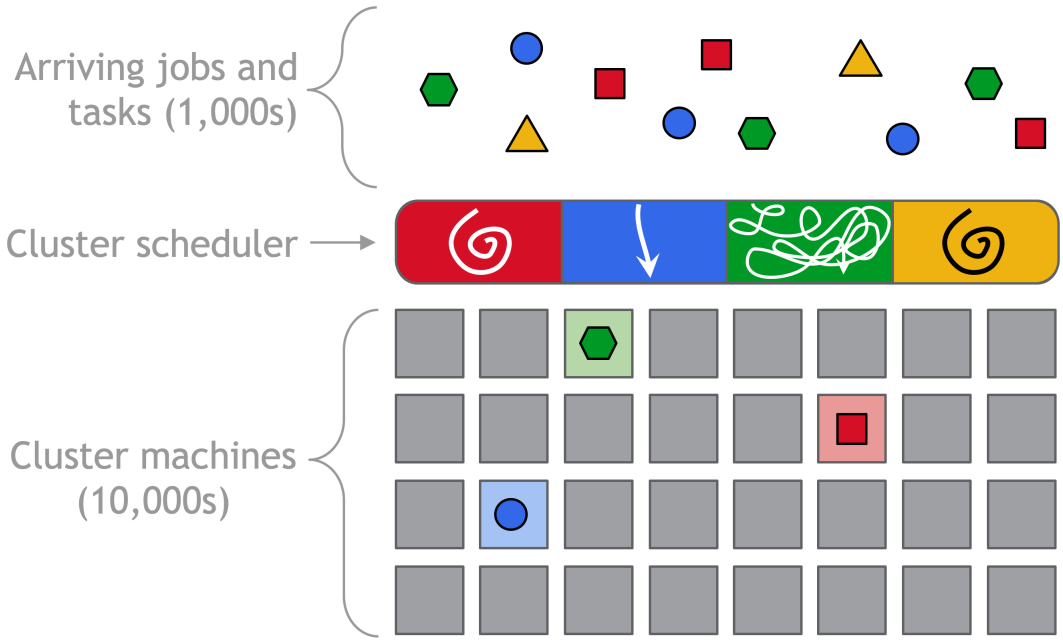

Рис. 1. Схема взаимодействия планировщика задач и кластера.

Планировщик, взаимодействующий с большим вычислительным кластером, должен учитывать множество факторов: приоритет, требования и объём задачи, текущая занятость ресурсов, политика выполнения и так далее. Всё это вместе с увеличением как самих кластеров, так и количества задач ведёт к усложнению логики, которой руководствуется планировщик, что сильно замедляет время как его работы, так и время выполнения самих задач.

Одним из возможных решением данной проблемы является замена одного планировщика на несколько, каждый из которых охватывает лишь определённую часть задач. Однако этим нескольким планировщикам всё ещё необходимо делить одни и те же ресурсы.

Разделение ресурсов

Прямолинейный подход заключается в фиксированном делении физического кластера, с которым работают несколько планировщиков, на столько же логических кластеров (подкластеров), тем самым выделяя каждому планировщику необходимые ресурсы.

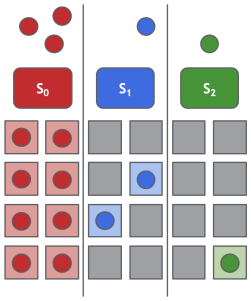

Рис. 2. Разделение кластера на логические подкластеры.

Проблемы такого подхода очевидны: допустим, на одном подкластере, соответствующем планировщику S_0, выполняются задачи MapReduce, а на остальных – задачи инфраструктурных сервисов, тогда первый подкластер полностью загружен, хотя в исходном кластере ещё есть свободные ресурсы под задачи S_0. Таким образом, данный подход может стать ограничением при попытках масштабировать систему.

Решением может выступить динамическое выделение ресурсов, для чего потребуется отдельный алгоритм и/или процесс, называемый **менеджером ресурсов**. Его работа заключается в предоставлении необходимых планировщикам ресурсов по мере поступления от них запросов, причём ресурсы, предлагаемые одному планировщику, недоступны для предложения другим – подобная логика реализована в Apache Mesos. Однако и у такого подхода есть недостатки.

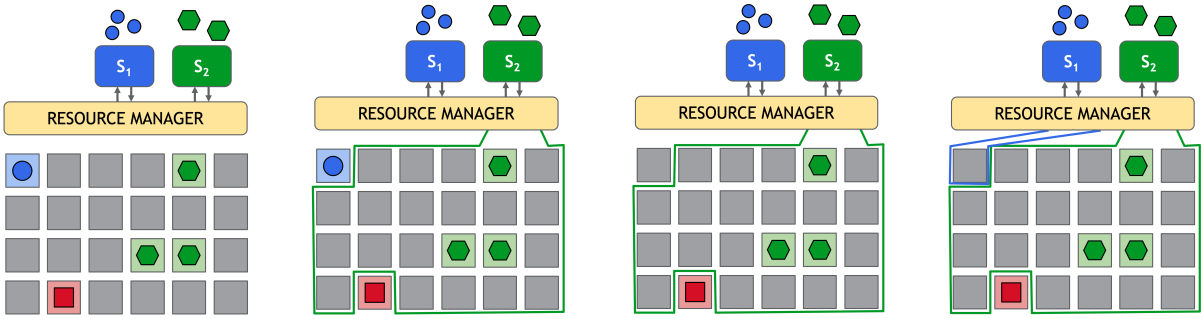

Рис. 3. Подход с менеджером ресурсов.

Предположим, что с кластером работают два планировщика S_1 и S_2, каждый из которых ждёт исполнения своих задач. Пусть S_2 устроен таким образом, что при поступлении новых задач он сразу же пытается запросить под них ресурсы. Менеджер ресурсов предлагает S_2 доступную часть кластера, однако для вычисления оптимального распределения задач на предоставленных ресурсах требуется время. Паралелльно с этим, S_1 дождался своих задач и запросил ресурсы под новые. В силу того, что большинство ресурсов было предложено планировщику S_2, а оставшихся не хватает для распределения задач S_1, последний ждёт предложения получше. Если планировщик S_2 будет рассчитывать оптимальное распределение задач слишком долго, S_1 рано или поздно сдастся и отменит выполнение задач, которые ему поступили, хотя ресурсов для них было предостаточно.

Этот подход ограничивает видимость ресурсов, а также нивелирует преимущества параллелизма работы планировщиков.

Система Omega

Возьмём в качестве основы предыдущий подход, но уберём из него менеджер ресурсов, тем самым разрешив планировщикам самим управлять ресурсами кластера (подробнее далее). Рассмотрим данный подход на примере.

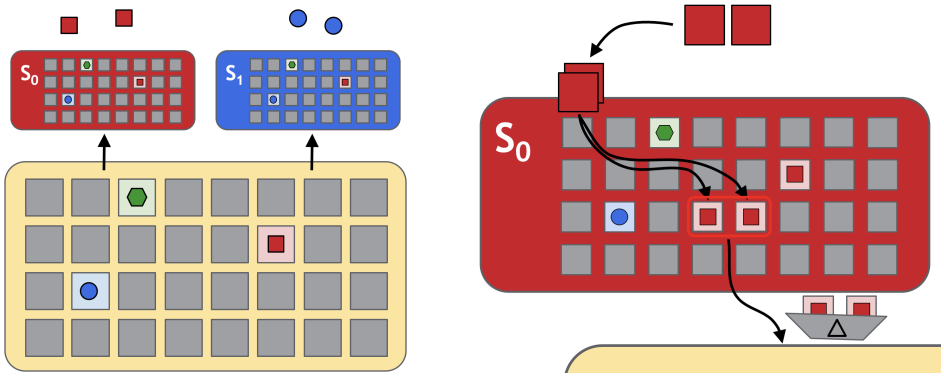

Рис. 4. S_0 отправляет кластеру запрос.

Положим, что каждый планировщик хранит у себя копию текущего состояния кластера, с которым идёт работа. При поступлении новых задач, планировщик, на основании своей копии, решает, как лучше всего распределить задачи, после чего посылает кластеру соответствующий запрос на изменение его состояния. Если конфликтов не возникло, то запрос одобряется и задачи считаются успешно распределёнными.

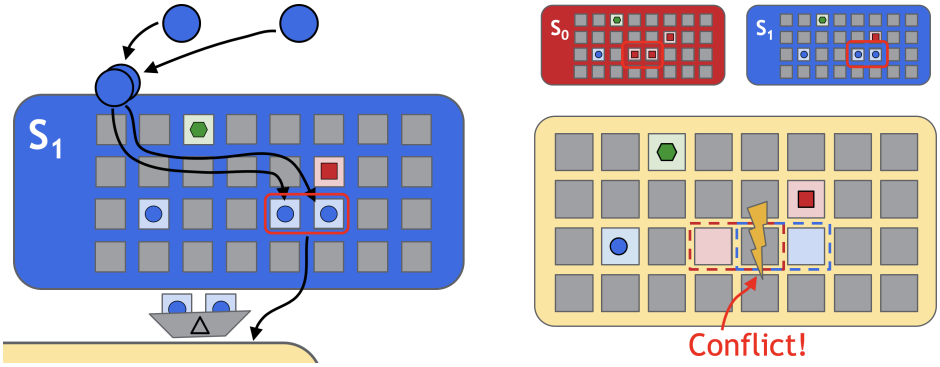

Рис. 5. Конфликт при параллельных запросах.

Теперь рассмотрим ситуацию, в которой планировщик S_1 параллельно с S_0 запросил те же ресурсы, что и S_0. При подобном конфликте, Omega, на основании определённых правил и политик выполнения, одобрит запрос только одному из планировщиков, отвечая остальным сообщением о неудаче.

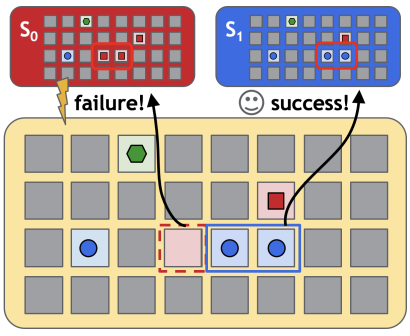

Рис. 6. Разрешение конфликта.

В таком случае, S_0 может попробовать отправить запрос ещё раз после того, как обновит локальную копию текущего состояния кластера. Понятно, что в худшем случае данному планировщику никогда не удастся заполучить требуемые ресурсы.

Таким образом, эффективность и производительность системы Omega зависит от частоты возникновения подобных конфликтов и способов их избежания.

Оптимизации

Рассмотрим несколько методов, позволяющих уменьшить вероятность конфликта:

1. каждой машине выдаётся счётчик использований (подобие логических часов Лампорта), который означает то, сколько раз планировщики обращались к данной машине, после чего при каждом запросе сначала сверяются актуальный счётчик и счётчик из локальной копии планировщика и в случае несовпадения сразу же отправляется сообщение о неудаче (что значительно быстрее непосредственных попыток обнаружить конфликт);
2. в случае конфликта, дать планировщикам возможность заполучать ресурсы не на все запрошенные задачи сразу, а только на часть из них (если задачи разделимы друг от друга), чтобы уменьшить объём запроса в следующий раз.

Данные оптимизации (как мы увидим далее) позволяют достичь значительных улучшений в работе системы Omega, достигая в некоторых случаях более чем двухкратного ускорения.

Результаты

Задачи, в некотором приближении данной статьи, делятся на два типа: сервисные (к примеру, поддержка инфраструктуры) и батчевые (например, вычисления MapReduce). Несмотря на то, что количество батчевых задач составляет более 90% от всех поступаемых планировщикам задач, большинство ресурсов уходит на поддержание сервисных задач, так как они требуют тщательного распределения ресурсов и выполняются крайне долго.

	Время выполнения	Частота поступления
Батчевые	12-20 минут	4-7 секунд
Сервисные	29 дней	2-15 минут

Таб. 1. Характеристики задач (80-й перцентиль).

Для замеров и сравнения результатов работы использовался самописный симулятор описанных выше подходов, который упрощал отдельно взятые моменты и использовал эмпирическое распределение для характеристик задач.

Время, которое планировщики тратят на распределение задач, является верхней границей того, что происходит на практике: как минимум 0.1 секунда необходима для самой задачи и 0.005 секунд для каждой подзадачи.

Соответственно, время, необходимое планировщику для того, чтобы решить, куда распределить задачи, выражается как $t_{\text{решения}} = t_{\text{задачи}} + t_{\text{подзадачи}} \times (\#\text{ подзадач})$.

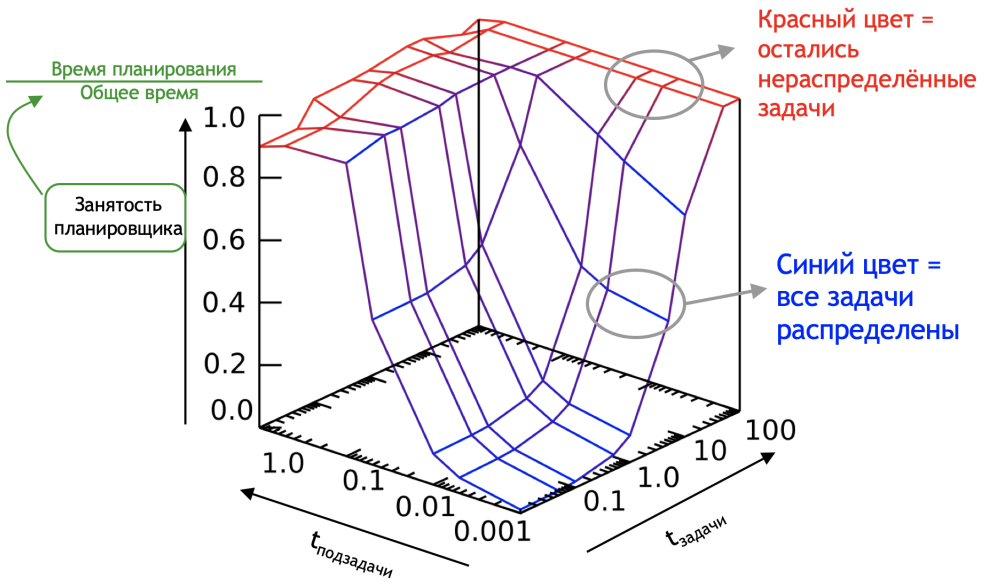

Рис. 7. Монолитный планировщик (шкалы в горизонтальной плоскости являются логарифмическими).

Базой/основанием/худшим результатом для сравнения выступает планировщик-монолит, который в одиночку распределяет все задачи по одному кластеру, руководствуясь схожей логикой для разных типов задач. Как видно, при увеличении времени, необходимого планировщику для принятия решения о распределении задач, сам планировщик перестаёт успевать распределять задачи, увеличивая общее время ожидания.

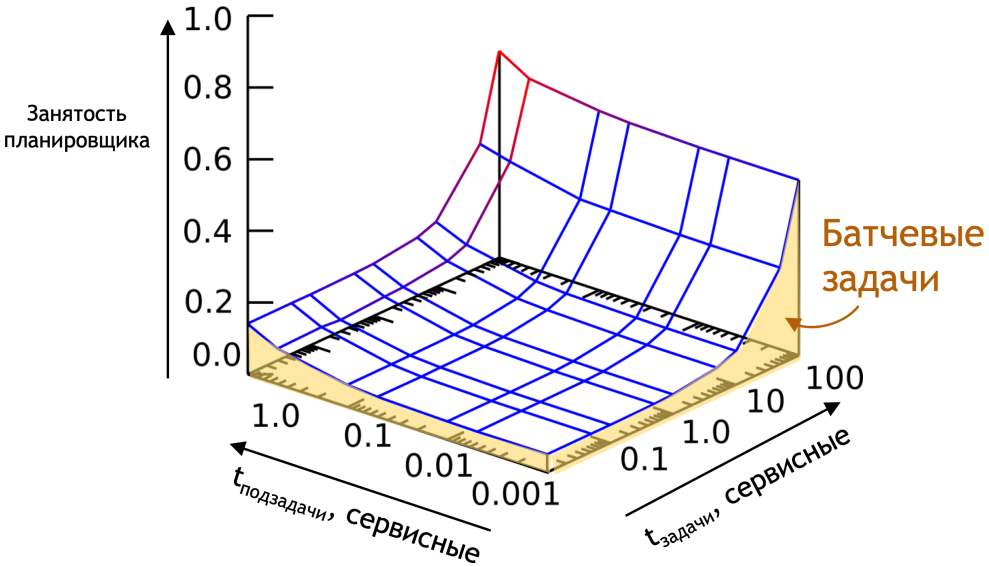

Рис. 8. Монолитный планировщик, разная логика для разных задач.

Следующим примером для сравнения выступает всё тот же планировщик-монолит, однако теперь он следует разной логике в зависимости от того, поступила ли ему сервисная задача, батчевая или какая-либо другая. Небольшой сдвиг относительно нуля, изображённый жёлтым цветом, отвечает за распределение батчевых задач, которые поступают относительно постоянно. Данный сдвиг присутствует в силу того, что монолитный планировщик не способен воспроизводить параллельные вычисления.

Здесь и всюду далее мы будем варьировать лишь время, необходимое для распределения сервисных задач.

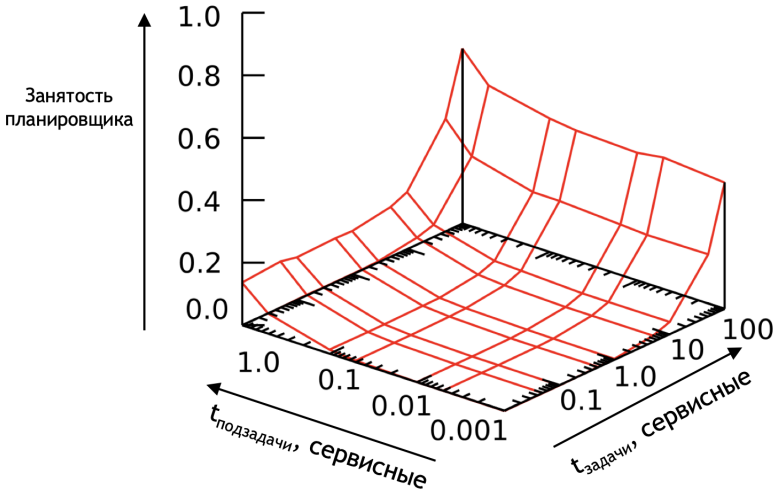

Рис. 9. Система Mesos.

Далее мы можем видеть результаты работы системы Mesos, которая использует менджеров ресурсов. Причина, по которой график раскрашен в красный цвет, была описана ранее – предлагаемые планировщику ресурсы недоступны для предложения остальным.

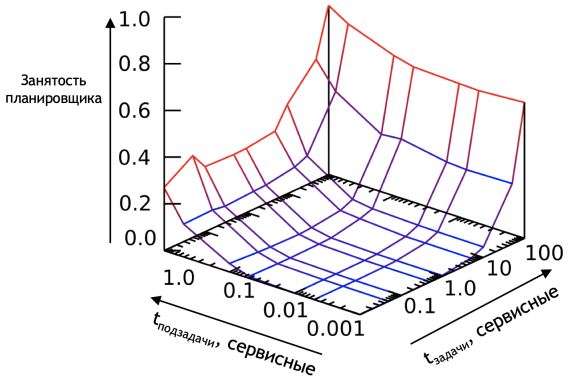

Рис. 10. Система Omega, без оптимизаций.

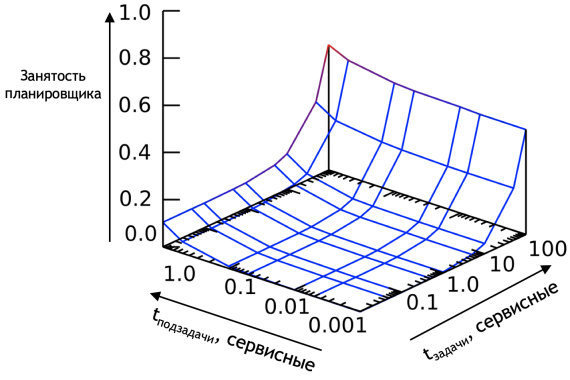

Рис. 11. Система Omega, с оптимизациями.

Наконец, рассмотрим систему Omega. Сдвиг относительно нуля здесь и у Mesos исчез в силу того, что планировщиков стало несколько и теперь они работают параллельно.

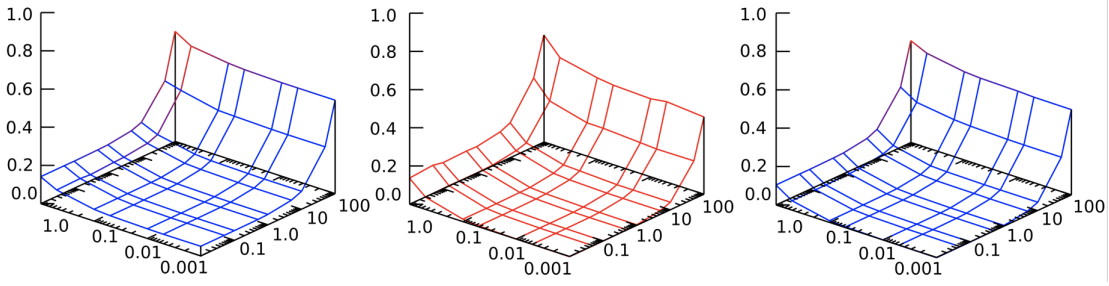

Рис. 12. Планировщик-монолит (слева), Mesos (по центру), Omega (справа).

В итоге Omega показывает себя так же хорошо, как и монолитный планировщик с разной логикой распределения для разных задач, а также не страдает проблемой Mesos, в которой даже при быстром распределении остаются нераспределённые задачи.

Масштабируемость

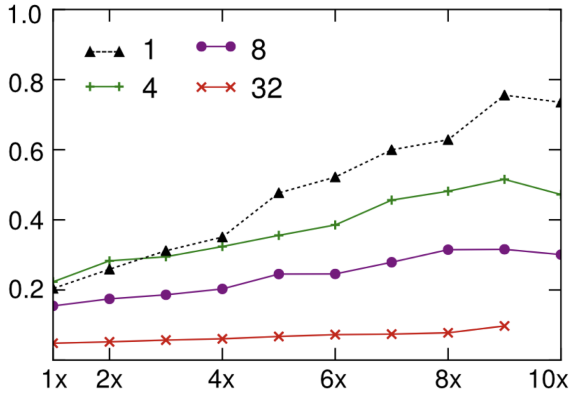

Рис. 13. Занятость планировщика в зависимости от количества задач.

На графике выше по вертикальной оси отложена занятость планировщика, а по горизонтальной оси – то, во сколько раз больше задач сейчас поступает планировщикам. Легко видеть, что при текущем подходе и оптимизациях, Omega хорошо масштабируется.

Кратко

Планировщик задач со временем накапливает множество правил и политик выполнения, которым он обязан следовать, однако это сильно замедляет общее время работы. Решением может послужить замена одного планировщика на нескольких, однако и тут возникают специфичные проблемы разделения ресурсов.

Известные архитектуры систем, работающих с планировщиками задач, и их недостатки сподвигли авторов на создание новой системы Omega, которая, хоть и зависит от количества конкурентных конфликтов, показывает себя достойно по сравнению с каждой из существующих систем и суммарно выигрывает у каждой из них, благодаря несложным оптимизациям.

Видимость общего состояния кластера всем работающим с ним планировщикам позволяет разрабатывать специфичные дизайны для планировщиков, которые были бы невозможны у того же монолитного планировщика или у Mesos.

Комментарии

Авторам удалось удачно позаимствовать идеи транзакций из мира баз данных, где подобные подходы давно известны и тщательно исследованы, и применить их в области планирования задач, что позволило ввести масштабируемость и параллелизм.

Кажется, что Omega всё же страдает несколькими проблемами: к примеру, небольшие кластеры с большим количеством запросов будут склонны к всё большему и большему увеличению конфликтов, что, в конце концов, способно значимо увеличить общее время работы планировщика и время ожидания распределения задач, что может повлиять на работоспособность всей системы.

Не совсем понятны: применимость Omega к не-Google системам (системам значительно меньше); соотношение количества конфликтов к работе, которую приходится делать заново

(расчёт оптимального времени и места распределения, отправка запросов, и так далее); работа с другими типами задач (помимо батчевых и сервисных).

Также было бы здорово увидеть исследование на тему справедливого распределения ресурсов и избежания ситуаций “голодания”, в которых какому-либо планировщику так и не удастся заполучить необходимые ресурсы.