Федеральное агентство связи (Россвязь)
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

010300 Фундаментальная информатика и
информационные технологии
№ кода и наименование направления подготовки

РАСЧЁТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

по дисциплине «Архитектура вычислительных систем»

Вариант № 01

Выполнил:
студент гр. ИС-441 _____ /Брагин А.С./
подпись

Проверил:
доцент кафедры ВС
к.т.н. _____ /А.В. Ефимов /
ОЦЕНКА, подпись

Новосибирск 2016

ОГЛАВЛЕНИЕ

1. ОТВЕТ НА ПЕРВЫЙ ВОПРОС

1.1. ЗАДАНИЕ

Осуществить анализ архитектуры мультипроцессорных вычислительных систем. Привести пример отечественной (промышленной) ВС.

1.2. ОТВЕТ

1.2.1. Мультипроцессоры

Различают два типа мультипроцессорных систем - multiprocessors (мультипроцессоры или системы с общей разделяемой памятью) и multicomputers (мультикомпьютеры или системы с распределенной памятью).

Первый вариант - использование единой (централизованной) общей памяти (shared memory) (см. рис. 1.2.1.1 а). Такой подход обеспечивает однородный доступ к памяти (uniform memory access или UMA) и служит основой для построения векторных параллельных процессоров (parallel vector processor или PVP) и симметричных мультипроцессоров (symmetric multiprocessor или SMP). Среди примеров первой группы - суперкомпьютер Cray T90, ко второй группе относятся IBM eServer, Sun StarFire, HP Superdome, SGI Origin и др.

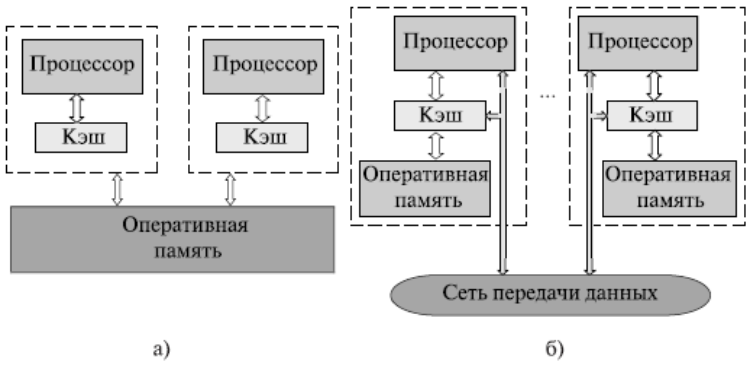

Рис. 1.2.1.1 Архитектура многопроцессорных систем с общей (разделяемой) памятью: системы с однородным (а) и неоднородным (б) доступом к памяти

Одной из основных проблем, которые возникают при организации параллельных вычислений на такого типа системах, является доступ с разных процессоров к общим данным и обеспечение, в связи с этим, однозначности (когерентности) содержимого разных кэшей (cache coherence problem). Дело в том, что при наличии общих данных копии значений одних и тех же переменных могут оказаться в кэшах разных процессоров. Если в такой ситуации (при наличии копий общих данных) один из процессоров выполнит изменение значения разделяемой переменной, то значения копий в кэшах других процессоров окажутся не соответствующими действительности и их использование приведет к некорректности вычислений. Обеспечение однозначности кэшей обычно реализуется на аппаратном уровне – для этого после изменения значения общей переменной все копии этой переменной в кэшах отмечаются как недействительные и последующий доступ к переменной потребует обязательного обращения к основной памяти. Следует отметить, что необходимость обеспечения когерентности приводит к некоторому снижению скорости вычислений и затрудняет создание систем с достаточно большим количеством процессоров.

Наличие общих данных при параллельных вычислениях приводит к необходимости синхронизации взаимодействия одновременно выполняемых потоков команд. Так, например, если изменение общих данных требует для своего выполнения некоторой последовательности действий, то необходимо обеспечить взаимоисключение (mutual exclusion), чтобы эти изменения в любой момент времени мог выполнять только один командный поток. Этим занимается программист. Задачи взаимоисключения и синхронизации относятся к числу классических проблем, и их рассмотрение при разработке параллельных программ является одним из основных вопросов параллельного программирования. Объектов синхронизации существует несколько, самые важные из них - это взаимоисключение (mutex), критическая секция (critical section), событие (event) и семафор (semaphore). Каждый из этих объектов реализует свой способ синхронизации. Также в качестве объектов синхронизации могут использоваться сами процессы и нити (когда одна нить

ждет завершения другой нити или процесса); а также файлы, коммуникационные устройства, консольный ввод и уведомления об изменении.

Общий доступ к данным может быть обеспечен и при физически распределенной памяти (при этом, естественно, длительность доступа уже не будет одинаковой для всех элементов памяти) (см. рис. 1.2.1.1 б). Такой подход именуется неоднородным доступом к памяти (non-uniform memory access или NUMA). Среди систем с таким типом памяти выделяют:

- системы, в которых для представления данных используется только локальная кэш-память имеющихся процессоров (cache-only memory architecture или COMA); примерами являются KSR-1 и DDM;
- системы, в которых обеспечивается когерентность локальных кэшей разных процессоров (cache-coherent NUMA или CC-NUMA); среди таких систем: SGI Origin 2000, Sun HPC 10000, IBM/Sequent NUMA-Q 2000;
- системы, в которых обеспечивается общий доступ к локальной памяти разных процессоров без поддержки на аппаратном уровне когерентности кэша (non-cache coherent NUMA или NCC-NUMA); например, система Cray T3E.

1.2.2. Мультикомпьютеры

Мультикомпьютеры (многопроцессорные системы с распределенной памятью) уже не обеспечивают общего доступа ко всей имеющейся в системах памяти (no-remote memory access или NORMA) (см. рис. 1.2.2.1). При всей схожести подобной архитектуры с системами с распределенной общей памятью (рис. 1.2.1.1 б), мультикомпьютеры имеют принципиальное отличие: каждый процессор системы может использовать только свою локальную память (кэш и оперативную память), в то время как для доступа к данным, располагаемым на других процессорах, необходимо явно выполнить операции передачи сообщений (message passing operations). Данный подход применяется при построении двух важных типов многопроцессорных вычислительных систем - массивно-параллельных систем (massively parallel processor или MPP) и

кластеров (clusters). Среди представителей первого типа систем - IBM RS/6000 SP2, Intel PARAGON, ASCI Red, транспьютерные системы Parsytec и др.; примерами кластеров являются, например, системы AC3 Velocity и NCSA NT Supercluster.

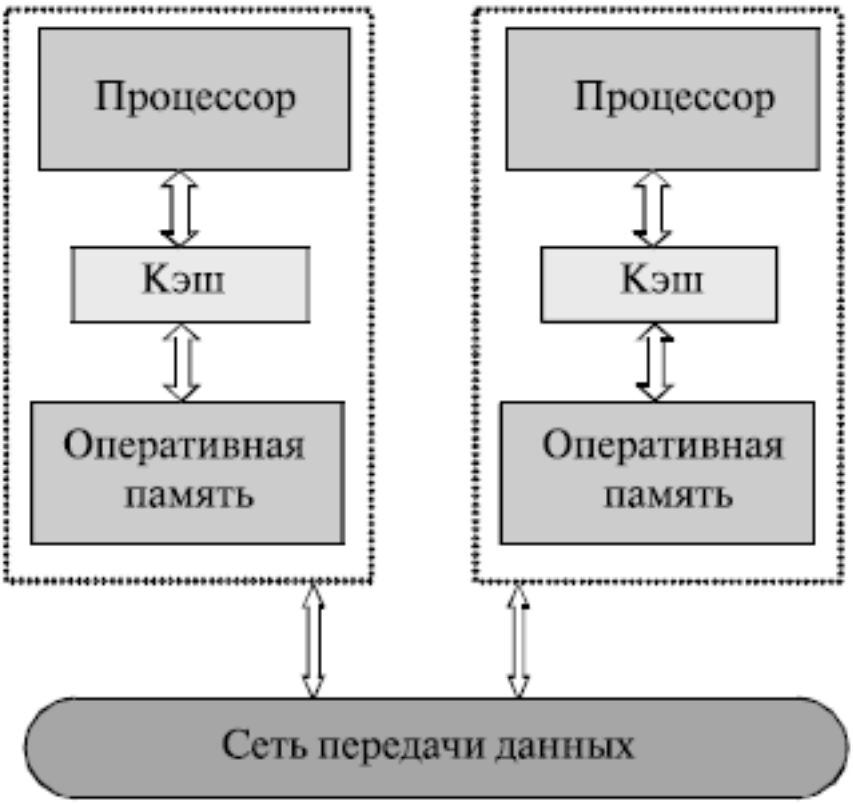

Рис. 1.2.2.1. Архитектура многопроцессорных систем с распределенной памятью

Следует отметить чрезвычайно быстрое развитие многопроцессорных вычислительных систем кластерного типа. Под кластером обычно понимается (см. [2, 3]) множество отдельных компьютеров, объединенных в сеть, для которых при помощи специальных аппаратно-программных средств обеспечивается возможность унифицированного управления (single system image), надежного функционирования (availability) и эффективного использования (performance). Кластеры могут быть образованы на базе уже существующих у потребителей отдельных компьютеров либо же сконструированы из типовых компьютерных элементов, что обычно не требует значительных финансовых затрат. Для создания кластера нужно получить набор

соединенных компьютеров. В идеале, машины должны быть максимально идентичными - так, чтобы ни одна из них не была "слабым звеном" во всей группе, что бы приводило к рассинхронизации и увеличению времени параллельного вычисления. В сети также должна быть однородность, ведь в большинстве случаев параллельные вычисления полагаются на постоянные соединения между всеми узлами кластера.

Следует отличать понятия кластера от сети компьютеров (network of workstations или NOW). Для построения локальной компьютерной сети, как правило, используют более простые сети передачи данных (порядка 100 Мбит/сек). Компьютеры сети обычно более рассредоточены, и пользователи могут применять их для выполнения каких-либо дополнительных работ.

Применение кластеров может также в некоторой степени устранить проблемы, связанные с разработкой параллельных алгоритмов и программ, поскольку повышение вычислительной мощности отдельных процессоров позволяет строить кластеры из сравнительно небольшого количества (несколько десятков) отдельных компьютеров (lowly parallel processing). Тем самым, для параллельного выполнения в алгоритмах решения вычислительных задач достаточно выделять только крупные независимые части расчетов (coarse granularity), что, в свою очередь, снижает сложность построения параллельных методов вычислений и уменьшает потоки передаваемых данных между компьютерами кластера. Вместе с этим следует отметить, что организация взаимодействия вычислительных узлов кластера при помощи передачи сообщений обычно приводит к значительным временным задержкам, и это накладывает дополнительные ограничения на тип разрабатываемых параллельных алгоритмов и программ.

В некоторых параллельных системах программирования передача данных между компонентами скрыта от программиста (например, с помощью механизма обещаний, где описывается объект, к которому можно обратиться за результатом, вычисление которого может быть не завершено на данный момент), тогда как в других она должна указываться явно. Явные взаимодействия могут быть разделены на два типа [см. 5].

- Взаимодействие через разделяемую память: на каждом процессоре мультипроцессорной системы запускается поток исполнения, который принадлежит одному процессу. Потоки обмениваются данными через общий для данного процесса участок памяти[2]. Количество потоков соответствует количеству процессоров. Потоки создаются либо средствами языка (например, в Java или C#, C++ (начиная с C++11), C (начиная с C11)), либо с помощью библиотек явно (например, в C/C++ с помощью PThreads), либо декларативно (например, с помощью библиотеки OpenMP), или автоматически встроенными средствами компилятора (например, High Performance Fortran). Данный вид параллельного программирования обычно требует какой-то формы захвата управления (мьютексы, семафоры, мониторы) для координации потоков между собой.
- Взаимодействие с помощью передачи сообщений: на каждом процессоре многопроцессорной системы запускается однопоточный процесс, который обменивается данными с другими процессами, работающими на других процессорах, с помощью сообщений. Процессы создаются явно, путём вызова соответствующей функции операционной системы, а обмен сообщениями — с помощью библиотеки (например, реализация протокола MPI), или с помощью средств языка (например, High Performance Fortran, Erlang или occam). Обмен сообщениями может происходить асинхронно, либо с использованием метода «рандеву», при котором отправитель блокирован до тех пор, пока его сообщение не будет доставлено. Асинхронная передача сообщений может быть надёжной (с гарантией доставки) либо ненадёжной.

Параллельные системы, основанные на обмене сообщениями, зачастую более просты для понимания, чем системы с разделяемой памятью, и обычно рассматриваются как более совершенный метод параллельного программирования. Существует большой выбор математических теорий для изучения и анализа систем с передачей сообщений, включая модель акторов и различные виды исчислений процессов. Обмен сообщениями может быть

эффективно реализован на симметричных мультипроцессорах как с разделяемой когерентной памятью, так и без неё.

У параллелизма с распределенной памятью и с передачей сообщений разные характеристики производительности. Обычно (но не всегда), накладные расходы памяти на процесс и времени на переключение задач у систем с передачей сообщений ниже, однако передача самих сообщений более накладна, чем вызовы процедур.

1.2.3. Пример отечественной (промышленной) ВС

Семейство вычислительных систем «Эльбрус». Работы по проектированию семейства ВС «Эльбрус» проводились в 1970-х и 1980-х годах под руководством В.С. Бурцева (1927-2005; академик РАН с 1992 г.) в Институте точной механики и вычислительной техники (ИТМиВТ) им. С.А. Лебедева АН СССР. Проект ВС «Эльбрус» был разработан в 1970 г., модель «Эльбрус-1» была принята Госкомиссией в 1980 г., а «Эльбрус-2» в 1985 г. [см. 6, 7]. Обе модели выпускались в СССР более 15 лет.

В рамках работ по проекту «Эльбрус» преследовалась цель создать семейство высокопроизводительных и надежных ВС. Для моделей семейства «Эльбрус» характерны:

- распределенное управление;
- однородность, модульность и масштабируемость структуры;
- надежность и самоконтроль;
- аппаратурная поддержка функций ОС и средств языка высокого уровня;
- разрядность слов 32, 64, 128;
- многоуровневая память;
- спецпроцессоры приема-передачи данных;
- производительность до 125 MFLOPS.

Любая из моделей семейства «Эльбрус» представляет собой распределенную вычислительную систему, построенную по модульному принципу. Система обладала свойством масштабируемости и допускала формирование

конфигураций. В состав системы «Эльбрус» могло входить от 1 до 10 ЦП, от 4 до 32 модулей оперативной памяти, от 1 до 4 процессоров ввода-вывода, от 1 до 16 процессоров приема-передачи данных, необходимое количество устройств внешней памяти и устройств ввода-вывода информации. Предусматривалась возможность подключения УВВ либо непосредственно к ПВВ, либо через каналы связи к ППД. Взаимодействие между подмножествами ЦП, МП и ПВВ осуществлялось через распределенный коммутатор, представлявший собой композит по локальных коммутаторов (ЛК).

Функционирование каждого из компонентов (ЦП, ПВВ, четырех модулей памяти и ЛК) системы «Эльбрус» поддерживалось средствами аппаратного контроля. Эти средства при появлении даже одиночной ошибки вырабатывали сигнал неисправности. По этому сигналу ОС через аппаратно-реализованную среду реконфигурации исключала неисправный компонент из рабочей конфигурации. Отключенный компонент попадал в ремонтную конфигурацию, где он при помощи контрольно-диагностических программ и специальной аппаратуры ремонтировался, после чего мог быть включен ОС в рабочую конфигурацию.

Средства реконфигурации позволяли организовать конфигурации ВС повышенной надежности. Так, например, допускались конфигурации ВС с резервом на уровне однотипных компонентов. Время включения резервного компонента в рабочую конфигурацию ВС не превышало 0,01 с. Надежность ВС выражалась в возможности передачи функций отказавшего компонента другому такого же типа, что, конечно, приводило к снижению рабочих характеристик (производительности, в частности), но не к отказу ВС в целом.

Помимо общедоступной оперативной памяти в системе «Эльбрус» имелась также и сверхоперативная память, распределенная по центральным процессорам. Каждое слово в ВС сопровождалось тегом, указывающим тип данных (целое, вещественное, адрес, метка процедуры и т. д.) и формат данных (32, 64 или 128 разрядов).

Работа ВС «Эльбрус» осуществлялась под управлением ОС. В этой ВС был реализован механизм управления вычислительными процессами, учитывающий

состав и состояние ресурсов ВС. Вычислительный процесс мог быть активным или пассивным. Одновременно несколько процессов могли быть активными, при этом любой процессор мог работать с любым из них. Для синхронизации процессов использовались семафоры; система команд содержала операции «открыть семафор» и «закрыть семафор». Если все процессоры были заняты работой, то могла образовываться очередь готовых к выполнению процессов. После освобождения какого-либо процессора (в частности, если реализуемый в нем активный процесс по тем или иным причинам становился пассивным или заканчивался) происходило обращение к очереди готовых процессоров и это т процессор начинал выполнять первый из них. Процессы могли перераспределяться в очереди в зависимости от их приоритетов [см. 4].

2. ОТВЕТ НА ВТОРОЙ ВОПРОС

2.1. ЗАДАНИЕ

Выполнить численный расчёт и построить график функции *r(t)* надёжности и коэффициента *s* готовности ЭВМ для следующих количественных характеристик:

- интенсивности отказов $\lambda = 10^{-3}$ 1/ч;
- интенсивности восстановления [?] = 1 1/ч.

2.2. ОТВЕТ

2.2.1. Функция надёжности

Функция (или вероятность безотказной работы) относится к основным показателям надежности ЭВМ. Характеризует производительность ЭВМ на промежутке времени, то есть эта функция обеспечивает потенциально возможную производительность. Функцией надежности ЭВМ называется

$$r(t) = P\{\forall\tau \in [0, t) \to \omega(\tau) = 1\}, \qquad (2.2.1.1)$$

где запись $P\{\forall\tau \in [0, t) \to \omega(\tau) = 1\}$ означает вероятность того, что для всякого τ, принадлежащего промежутку времени $[0, t),$ производительность $\omega(\tau)$ ЭВМ равна единице, т.е равна потенциально возможной.

Функция $r(t)$ обладает следующими свойствами:

- $r(0) = 1;$ Т.е. машина в момент начала функционирования находится в работоспособном состоянии.
- $r(+\infty) = 0;$ Событие, заключающееся в том, что ЭВМ работоспособна на конечном промежутке времени, является достоверным.
- $r(t_1) \geq r(t_2)$ для $t_1 \leq t_2$

Функцией ненадежности(или вероятностью отказа) ЭВМ называется

$$q(t) = 1 - r(t). \qquad (2.2.1.2)$$

Функция $q(t)$ позволяет определить среднее время безотказной работы (средняя наработка до отказа). По определению, *среднее время* ϑ *безотказной работы* ЭВМ и оценка $\tilde{\vartheta}$ соответственно равны:

$$\vartheta = \int_0^\infty t dq(t) = -\int_0^\infty t dr(t) = -tr(t)\Big|_0^\infty + \int_0^\infty r(t)dt = \int_0^\infty r(t)dt; \qquad \tilde{\vartheta} = \frac{1}{N}\sum_{i=1}^{N} t_i, \qquad (2.2.1.3)$$

где t_i – время безотказной работы i-й машины, $i \in \{1, 2, ..., N\}.$

Интенсивностью отказов (лямбда-характеристикой) ЭВМ называется функция

$$\lambda(t) = \frac{1}{1 - q(t)} \frac{dq(t)}{dt} = -\frac{1}{r(t)} \frac{dr(t)}{dt}. \qquad (2.2.1.4)$$

Практически установлено, что зависимость интенсивности отказов от времени имеет место на периоде приработки ЭВМ. После приработки ЭВМ интенсивность отказов остается постоянной (до вхождения в предельное состояние или, по крайней мере, в течение промежутка времени, перекрывающего время морального старения). Следовательно, в нормальных

условиях эксплуатации ЭВМ $\lambda = const$, а функция надежности и математическое ожидание времени *безотказной работы* соответственно равны:

$$r(t) = \exp(-\lambda t); \quad \vartheta = \int_0^\infty e^{-\lambda t} dt = -\frac{1}{\lambda} e^{-\lambda t} \Big/_0^\infty = \frac{1}{\lambda} \qquad (2.2.1.5)$$

λ – среднее число отказов, появляющихся в машине в единицу времени.

Подставляя известные нам данные получим следующую функцию для расчета надежности:

$r(t)=\exp(-\lambda * t)$.

$r(t)=\exp(-0.001*t)$.

Рассчитаем значения функции и построим график:

t, ч.	*r(t)*
0	1
1	0.999000499833375
5	0.995012479192682
10	0.990049833749168
100	0.90483741803596
1000	0.367879441171442
10000	0.0000453999297624849
20000	0.000000020611536224385 6
30000	0.000000000000935762296884017
40000	0.000000000000000042483542552915 9
50000	0.000000000000000000001928749847963 92
60000	0.000000000000000000000000875651076269652
70000	0.000000000000000000000000000039754497359086 5
80000	0.000000000000000000000000000000018048513878454 2
90000	0.000000000000000000000000000000000000819401262399051
100000	0.000000000000000000000000000000000000000037200759760208 4

Таблица. 2.2.1.1. Расчёт функции *r(t)*.

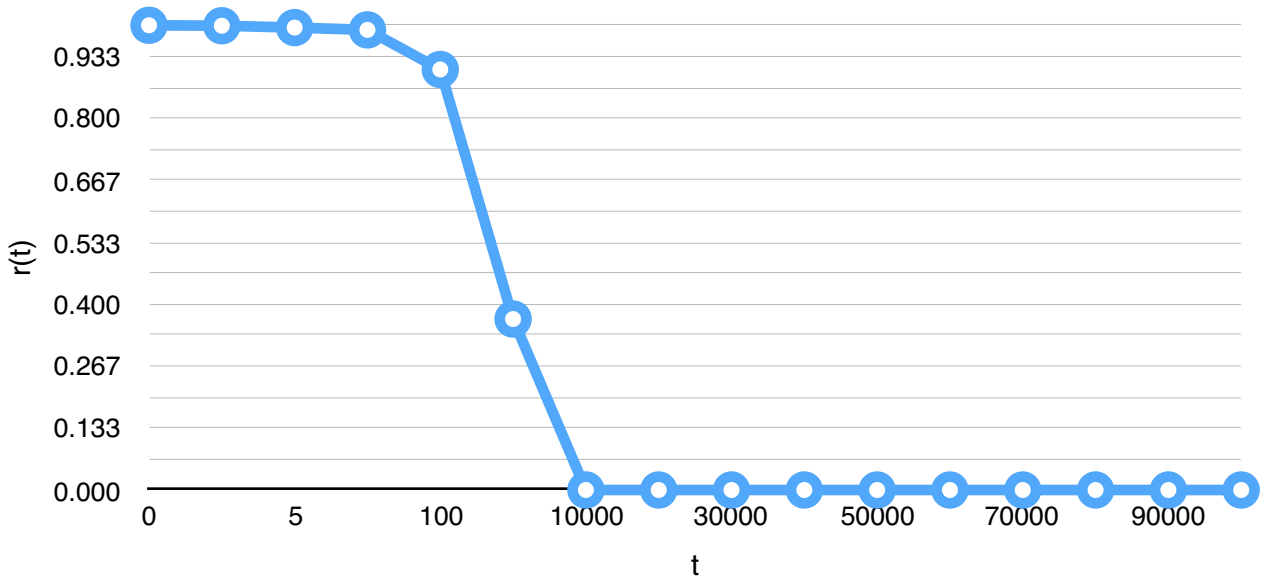

Рис. 2.2.1.1 График функции *r(t)*.

2.2.2. Коэффициент готовности

Функция готовности ЭВМ

$$s(i,t) = P_1(i,t) = P\{i;\ \ \omega(t) = 1\}, \tag{2.2.2.1}$$

$P\{i;\ \ \omega(t) = 1\}$, есть вероятность того, что (в условиях потока отказов и восстановлений) машина будет иметь в момент времени $t \geq 0$ производительность, равную единице, т.е. равную потенциально возможной.

Функция готовности ЭВМ обладает следующими свойствами:

- $s(0,0) = 0,\ s(1,0) = 1;$
- $s(i,\ +\infty) = s = const,\quad o < s < 1,\quad i \in E_0^1;$
- $s(0,t_1) \leq s(0,t_2),\ \ s(1,t_1) \geq s(1,\ t_2)$ для $t_1 \leq t_2.$

Расчёт будем производить по следующим формулам:

$$s(0,t) = \frac{\mu}{\lambda+\mu} - \frac{\mu}{\lambda+\mu} e^{-(\lambda+\mu)\cdot t} \tag{2.2.2.2}$$

$$s(1,t) = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{\lambda+\mu} e^{-(\lambda+\mu)\cdot t} \tag{2.2.2.3}$$

для начальных состояний ЭВМ $i = 0,\ i = 1$, причем $i = 0$ соответствует состоянию отказа, а $i = 1$ – работоспособному состоянию машины, где $\lambda = 0.001$.

s(0,t) = (1 / 1.001) - (1 / 1.001) * exp(1.001 * (-t)) = 0.999001 - 0.999001 * exp(1.001 * (-t));

s(1,t) = (1 / 1.001) + (0.001 / 1.001) * exp(1.001 * (-t)) = 0.999001 + 0.000999 * exp(1.001 * (-t));

Рассчитаем значения функции и построим график:

t, ч.	*s(0,t)*
0	0
0.001	0.00099949966795787
0.01	0.00995011659303413
0.08	0.0768806194247144
0.1	0.0951579033736871
0.3	0.259144846863541
0.5	0.393379151056241
1	0.63185639862517
10	0.998956096710331

Таблица. 2.2.2.1. Расчет функции *s(0,t)*.

t, ч.	*s(1,t)*
0	1
0.001	0.999999000501333
0.01	0.999990049893367
0.08	0.999923119457533
0.1	0.999904842191879
0.3	0.99974085541254
0.5	0.999606621242716
0.9	0.999406797707747
1	0.999368144233863
10	0.999001044903245

Таблица. 2.2.2.2. Расчет функции *s(1,t)*.

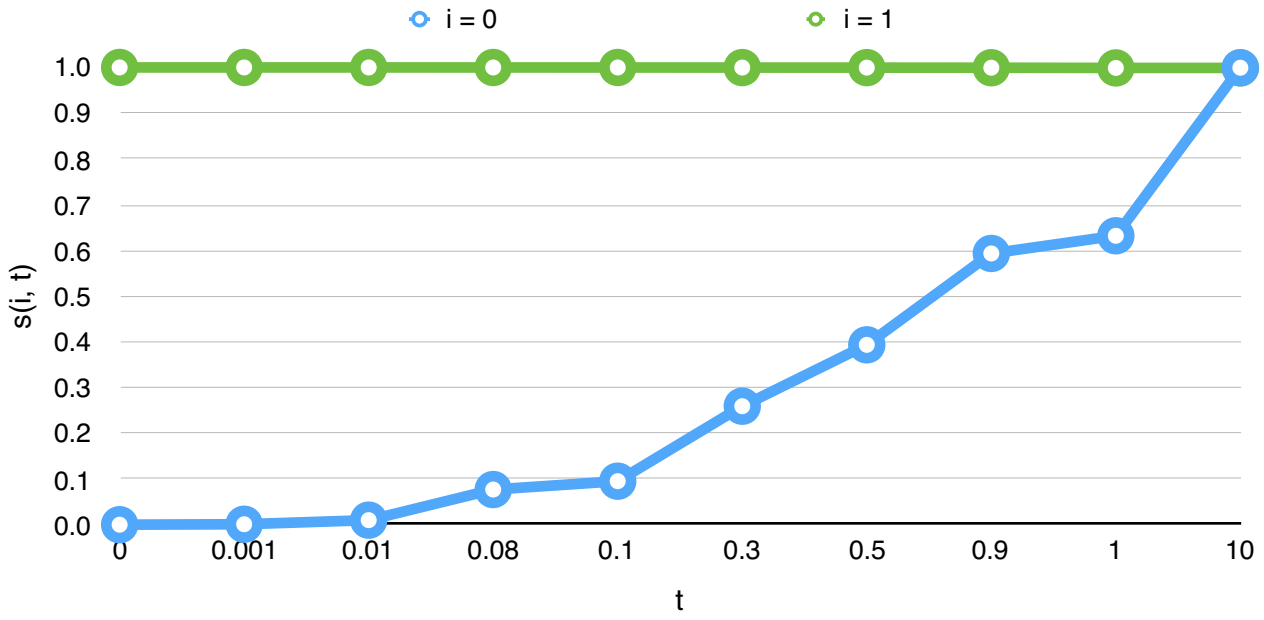

Рис. 2.2.2.1. График функции *s(i,t)*

3. СПИСОК ЛИТЕРАТУРЫ

1. Интернет-Университет Суперкомпьютерных Технологий. Теория и практика парралельных вычислений. http://www.intuit.ru/studies/courses/1156/190/lecture/4942?page=4
2. Pfister G. P. In Search of Clusters. Prentice Hall PTR, Upper Saddle River, NJ, 1995 (2nd edn., 1998).
3. Hwang K, Xu Z. Scalable Parallel Computing Technology, Architecture, Programming. Boston: McGraw-Hill, 1998.
4. Хорошевский В.Г., Архитектура вычислительных систем. – Н.: СибГУТИ, 2000.
5. Википедия. Параллельные вычисления. https://ru.wikipedia.org/wiki/Параллельные_вычисления
6. Смирнов А.Д. Архитектура вычислительных систем. М.: Наука, 1990.
7. Головкин Б.А. Параллельные вычислительные системы. М.: Наука, 1980.