워크플로우 엔진을 위한 태스크할당 알고리즘의 설계

이 헌[0]　　박규석
경남대학교　컴퓨터공학과
hlee@mail.koje.ac.kr, kspark@hanma.kyungnam.ac.kr

Design of Task Allocation Algorithm for Workflow Engine

Heon Lee[0]　　Kyoo-Seok Park
Dept. of Computer Engineering, Kyung Nam University

요 약

워크플로우 시스템에서 중요한 기능은 정의된 규칙, 절차 및 조건에 의한 작업흐름을 자동화하는 것으로 정보처리의 이동성을 고려하여야 한다. 특히, 최근에는 웹 환경의 확대에 따라 분산시스템에 대한 관심이 높으며, 분산시스템에서의 워크플로우 프로세스는 다른 노드에서도 태스크 처리가 가능해야 하고 실시간 처리가 요구된다. 또한 분산시스템을 구성하는 노드에서의 결함 발생은 전체 작업처리 효율을 저하시키므로 결함이 허용되는 분산 시스템에 적합한 워크플로우 엔진이 필요하다.

본 논문은 워크플로우 시스템이 분산시스템 환경에서 수행될 때, 결함을 허용하며 작업을 자동으로 처리하여 전체 프로세스의 처리율을 향상시킬 수 있는 태스크 할당 알고리즘을 설계하고, 태스크의 수행 보증율을 통해 분석한다.

1. 서 론

WfMS(Workflow Management System)은 조직과 조직이 수행하는 업무, 작업자간의 협업 도모, 작업의 자동화 및 일관적인 접근과 제어 등으로, 적용 분야는 보험회사의 사고처리업무, 금융회사의 융자업무, 프로젝트 진행업무, 시스템 관리와 예외처리, 신제품 개발 업무 및 병원연계 가정 건강관리 등이 있으며 관련 연구가 활발히 진행 중이다.

특히, 분산워크플로우 시스템을 적용하기 위한 연구는 수행 프로세스가 공통으로 처리될 수 있거나 복수의 서버에서 수행될 경우 일정 수준의 신뢰성과 가용성을 제공하기 위한 결함허용 컴퓨팅시스템 구축이 필요하다[1][2][3]

본 논문에서는 이러한 특성을 고려하여 분산워크플로우 시스템의 워크플로우 엔진이 필요로 하는 프로세스의 태스크 할당 알고리즘을 제안하였다. 제안 시스템은 작업흐름을 프로세스 별로 나누어 태스크를 산출한 후 절차적으로 수행하는 프로세스 엔진과, 서버노드의 결함허용을 지원하는 전처리기, 태스크의 서비스 서버를 선정하고 보증여부의 판정을 위한 태스크 스케줄러 등으로 구성되며, 전체 시스템의 가용성과 신뢰성을 향상시킨다.

2. 관련 연구

워크플로우 관리시스템은 업무 흐름의 자동화와 정보 및 문서 전달의 전자화, 그리고 일관적인 데이터 접근과 제어를 통해 업무 프로세스의 개선, 통제, 관리 및 공동작업을 지원하는 소프트웨어로서 문서보다는 과정을 중시한다[4][5][6].

이와 관련한 표준화로는 WfMC(Workflow Management Coalition)의 5개 인터페이스 및 메타모델이 있고[3], PIF(Process Interchange Format) Work Group은 프로세스 정보교환 형식에 대한 표준을 연구하고 있으며[7], NIST(National Institute of Standards and Technology)는 프로세스 표현언어에 대한 필요사항을 명세하였다[8].

결함허용을 위한 연구로는 웹 서비스와 연계하여 결함발생시 웹 서버의 부하 분산기에 의한 클러스터링 시스템을 구축하고, 클러스터 내에 있는 특정 서버에 부하를 집중시키지 않도록 하는 부하 분산방법이 있다. 부하분배 스케줄링 방법으로는 라운드 로빈 스케줄링, 가중 라운드 로빈 스케줄링, 연결된 사용자 요구 수가 제일 적은 곳을 우선적으로 할당하는 최소 연결 스케줄링 및 실시간 환경에 적합한 데드라인 우선 정책 등이 있다[9].

3. 시스템 모델

분산워크플로우 시스템 모델은 그림 1과 같다. 워크플로우 컨트롤 데이터는 다수 개의 워크플로우 엔진을 통제하기 위하여 해당 정보를 관리하는 곳으로 전체 분산 워크플로우 시스템의 주요 정보이다.

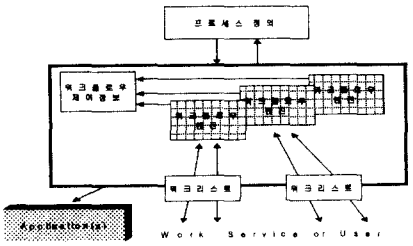

그림 1. 분산 워크플로우 시스템 모델

3.1 처리 모델

정보전달 및 수행 모델은 그림 2와 같이 나타낼 수 있다. 이것의 기능을 확장하여 분산실시간 시스템에 적용할 수 있도록 하는 경우 각 서버노드마다 자신이 지니고 있는 DB를 이용하고 상호간에는 메일을 이용한 메시지를 통해 작업흐름을 제어할 수 있다.

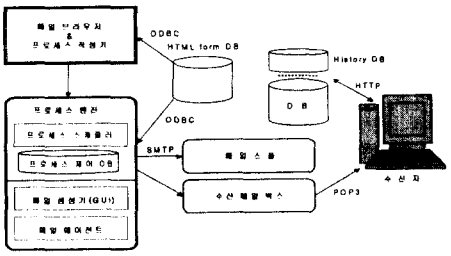

그림 2. 워크플로우 수행모델

메일 브라우저는 정보전달 및 상호 동작을 위하여 SMTP 프로토콜을 이용하고, 전송된 정보를 획득하기 위하여 POP3 프로토콜을 지원한다.

프로세스 엔진은 프로세스 제어를 위한 DB와 프로세스 스케줄러로 구성되고, 데이터베이스 구동방식과 메일 구동방식을 혼용하며, 수행 서버 선정을 위해 태스크 스케줄링을 한다. 프로세스 제어 DB는 프로세스 스케줄링과 태스크 스케줄링에 필요한 정보를 제공하고 수행상태를 모니터할 수 있게 하며 각 서버노드의 상태를 파악한다.

3.2 프로세스 엔진

프로세스 엔진의 역할은 업무 정의 해석, 프로세스 인스턴스 생성 및 시작, 종결, 일시정지, 재시작 등의 실행관리, 태스크 종결시의 프로세스 상태 수정, 태스크서비스에게 업무 전달, 프로세스 진행 과정상의 항목을 생성하고 태스크간의 이동제어, 태스크나 프로세스가 데드라인을 넘겼을 때의 조치 및 진행 중인 프로세스에 대한 감독과 관리 등이다.

프로세스 제어를 위해 전처리기와 태스크 스케줄러로 구성된 프로세스 스케줄러를 지니며 스케줄 정보를 받아들이기 위해 데이터베이스와 연계된다. 프로세스 엔진의 구성도는 그림 3과 같다.

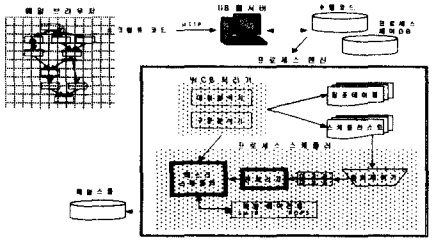

그림 3. 프로세스 엔진 구성도

WCS(Workflow Control Script) 처리기는 수행코드 데이터베이스에서 프로세스 흐름 제어 스크립트 코드의 어휘 및 구문분석을 수행하고, 태스크 스케줄 정보 테이블과 태스크 수행 상태 테이블을 생성하며 스케줄 태스크 테이블에 수행코드를 등록한다. 언어 분석기에서 의미분석을 마치면 흐름 제어 코드마다 태스크 스케줄러의 쓰레드를 할당받아 스케줄 정보로서 관리하며, 생명주기에 의해 상태를 변경한다.

프로세스 스케줄러의 구성은 클럭 제어기, 메일 에이전트, 전처리기, 태스크 스케줄러 및 관련 정보를 저장하는 공간 등이다. 대기 큐는 스케줄 대기 큐와 응답 대기 큐가 있으며, 클럭 제어기는 시간 알람을 이용하여 작업 제어 리스트를 스케줄 대기 큐로 보내고, 대기 큐의 내용은 태스크 스케줄러 또는 전처리기에서 사용한다.

프로세스 유형은 다음 두 가지 특징을 갖는다. 서로 다른 프로세스 엔진에 의해 관리되는 단위 작업들이 불규칙적으로 혼합되어 하나의 복합된 프로세스를 이루는 것과, 하나의 단위 작업이 다른 엔진에서 작동되는 여러 하부 업무 프로세스를 생성시키는 것이다. 수행은 프로세스의 태스크를 직접 실행하여야 하는 경우와 단순히 참조하는 경우로 구분할 수 있고, 피드백하는 경우도 포함된다. 또한 태스크는 지정된 서버노드에서 실행되어야 하는 엄격한 지역 태스크와 전체 또는 일부 그룹에서 공통으로 실행될 수 있는 전역 태스크가 있다.

프로세스 스케줄러는 결함허용을 위해 전처리기를 두며, 결함 발생시 지역 태스크의 경우에는 특별히 지명된 대리 사용자가 처리할 수 있거나 지연되고, 전역 태스크는 부하 균형을 고려하여 실행 노드를 신속히 선정해야 한다. 모든 프로세스는 실시간으로 처리를 요구하고 해당 데드라인을 만족시키지 못하면 다시 제출되거나 거부된다.

전처리기와 태스크 스케줄러의 관계는 그림 4와 같다. 클럭 제어기(Clock Controller)는 WCS 처리기가 생성한 스케줄 리스트에서 지정된 시간에 수행될 스케줄 리스트를 추출하여 스케줄 대기 큐에 저장하며, 태스크 스케줄러 또는 전처리기와 상호 협조적으로 동작한다. 태스크 스케줄러에서 수행된 스케줄에서 데드라인을 가지는 태스크는 데드라인 정보가 스케줄 상태 테이블, 스케줄 리스트 테이블을 이용하여 클럭 제어기의 스케줄 대기 큐에 등록된다.

전처리기(Preprocessor)는 도착한 태스크와 그에 대응하는 정보 DB 및 프로세스 제어 DB 등을 이용하여 해당 태스크의 보증을 시도한다.
태스크는 주기적으로 발생한 것과 비주기적으로 발생한 것을 구분하고, 주기적 태스크는 곧장 수행을 위한 판정을 위해 지역 스케줄러로 보낸다. 단, 서버노드나 웹 상의 결함(fault)이 발생한 경우에는 데드라인과 서비스 시간을 고려하여 긴급하지 않는 경우, 태스크 진행 과정을 History DB에 보관시키고 해당 메시지를 발생시켜 실행을 기다리도록 한다. 실행이 긴급한 것 중에서 공통 태스크는 전체 시스템에서 해당 클러스터 정보를 참조하여 수행 노드로 전송을 준비한다. 이 과정에서는 정밀한 노드 선정이 되지 못하므로 전역 스케줄러에서 보증을 확인 받아야 하며, 그 순간까지 작성된 정보를 메일과 메시지를 통하여 공유할 수 있도록 제공한다.

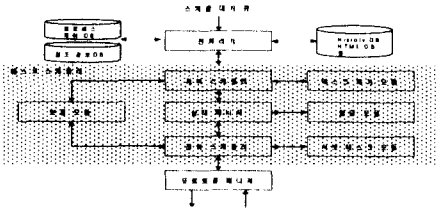

그림 4. 전처리기와 스케줄러 구조

다른 서버노드로부터 실행 전송된 전역 태스크나 결함으로 인한 대리 수행 요구 태스크는 전처리기가 관리하고 있는 factor를 참조하여 보증을 시도한다. 이 때 전처리기의 정보는 프로세스 엔진의 수행 대기 큐가 변경될 때마다 factor 값을 산출하여 갱신시킨다. 따라서 factor 값에 의한 보증 판정은 엄격하지 못하나 신속성과 오버헤드 측면에서 유리하며, 업무에서 분산워크플로우 시스템이 반복적으로 사용되는 경우 그 History를 통해 경험적 정보 누적으로 엄격성을 높일 수 있다.
결함 허용(Fault-Tolerant)을 위해 공유(share)노드, 중복(spare)노드 식별자 등에 대한 정보를 관리한다. 단, 전처리기가 지니고 있는 정보는 스케줄링을 신속히 하기 위한 최소 정보이며, 상세한 정보는 프로세스 제어 DB에서 참조한다.

태스크 스케줄러에서의 태스크 관리를 위한 정보는 다음과 같다.

T_id	P_id	Pri	J_ty	D_ln	Sv_t	St_t	Ag_n	Pe_c	C_id

그림 5. 태스크 테이블 구조

T_id는 태스크 식별자, P_id는 프로세스 식별자이며, Pri는 태스크의 우선 순위, J_ty는 태스크의 타입을 나타낸다. 그리고 D_ln은 태스크의 데드라인, Sv_t는 서비스 시간, St_t 는 시작 시간이고 Ag_n은 대리인을, Pe_c는 태스크의 주기성 여부를 나타내며 C_id는 클러스터 식별자로 태스크의 분리 가능여부이다. 이 외에도 태스크 이름, 태스크 기술내용, 문서 이름, Due time, Split 타입, Merge 타입, Role 이름, Abort 가능여부, Reject 가능여부, 제어 가능여부 등이 포함된다. 서비스 시간은 프로세스 정의에 의해 설정되고 History DB로부터 갱신될 수 있도록 하여 경험적 작업 흐름이 가능하다. 태스크의 종류는 Regular, Dummy, Irregular 및 Agent가 있으며, Split타입은 AND, OR와 조건에 따라 흐름이 변경되는 CONTROL 및 default 값인 NONE을 둔다. Merge 타입은 Split 타입과 동일하게 적용된다.

태스크 보증에 필요한 제어변수와 조건 정보는 다음과 같다.

RT_id	CV_id	CV_nm	CV_dc	CV_vl	CV_ty

그림 6. 제어변수 테이블

제어변수 테이블은 관계된 태스크, 제어변수 식별자, 제어변수명, 제어변수 내용, 제어변수 값 및 타입 등으로 구성된다.

지역 스케줄러는 전처리기로부터 도착한 프로세스에서 해당되는 태스크를 분리하여 실행 보증을 위해 보증 모듈을 호출한다.

보증 모듈은 태스크 테이블을 이용하여 세부적인 실행 여부를 판정하기 위해 데드라인을 참고한다. 워크플로우 시스템에서 발생하는 프로세스는 긴급정도별 데드라인을 지니고 있는 실시간 시스템을 전제로 한다. 따라서 보증 모듈에서는 먼저 태스크의 주기성을 분석하고 태스크가 지니고 있는 데드라인 우선 정책과 surplus를 이용하여 실행 보증여부를 판단한 후, 보증될 경우 보증 수행 대기 태스크들을 고려하여 실행 스케줄을 다시 작성하여 해당 정보를 보관하고 시작시간을 예약시킨다. 만약 프로세스나 태스크에 우선 순위(priority)가 있는 경우 surplus를 늘리기 위해 태스크 제거 모듈을 호출하여 제거 태스크를 정하고 surplus를 재 산출한 후 스케줄링을 행한다.

surplus 산출 방법은 식(1)과 같다.

$$surplus = Wt - \sum_{i=1}^{n}(Pt_i(Wt/P)) - \sum_{j=1}^{m}(Gt_j) \quad \cdots (1)$$

여기서 Wt는 작업가능 시간, Pti는 주기적 태스크의 서비스 시간, P는 주기적 태스크의 주기, Gtj는 주기적 태스크를 제외한 보증 태스크의 서비스 시간이다. n과 m은 각각 Wt 내에서의 주기적 태스크의 개수 및 보증된 비주기적 태스크의 개수를 나타낸다.

태스크 제거 모듈은 보증된 태스크들 중에서 우선 순위와 데드라인을 고려하여 제거 태스크를 선정하고 새로이 보증될 태스크의 서비스 시간을 반영시킨다. 여기서 제거된 태스크와 보증 모듈에서 실행이 보증되지 못한 태스크는 프로세스 제어 DB를 참조하여 전역 태스크 여부를 식별하고, 전역 태스크인 경우에는 태스크 제거 모듈을 통해 지역 스케줄러로부터 제거하여 전역 스케줄러로 보낸다. 전역 태스크가 아닌 경우에는 대기 큐에 넣어 다시 스케줄링에 참여될 수 있게 한다.

전역 스케줄러는 지역 스케줄러에서 보증되지 못한 태스크들 중 전역 태스크 또는 결함허용을 위한 대리 서비스(spare) 지정에 의한 태스크 등을 보증한다. 여기서도 보증되지 못한 태스크는 다시 서브 태스크 모듈로 보내어져 다수의 서브 태스크를 구한다. 만약 서브 태스크를 작성할 수 있다면 하나 또는 그 이상의 노드에서 보증될 수 있다.

상태 매니저는 지역 및 전역 스케줄러에서 보증된 태스크의 모든 정보를 관리하고, 서브 태스크를 확인하여 스케줄링 과정에서 참조되게 한다.

태스크 제거 모듈은 지역 스케줄러가 도착한 태스크를 보증할 수 없을 때 호출된다. 태스크 제거 모듈은 수행 대기 큐에서 디스패쳐 되기를 기다리는 비주기 태스크를 제거하여 surplus를 증가시켜 도착한 태스크를 보증하려 하는 것으로 도착한 태스크보다 데드라인 시간이 늦은 태스크를 제거하는 정책을 사용한다.

4. 분석

제안 시스템은 서버 시스템으로 Micro Soft Windows NT를 기반으로 IIS 4.0 웹 서버와 MS-SQL 6.5를 데이터베이스로 사용하였다. 클라이언트 시스템은 Windows 98 기반으로 메일 브라우저를 구현하였고, 데이터 흐름 제어 스크립트는 ASP 프로그램으로 웹 서버에 등록하도록 하였으며, V C++ 언어로 메일 브라우저와 프로세스 엔진을 설계하였다. 시뮬레이션에서의 함수는 SMPL을 이용하여 생성시켰다.

시뮬레이션 모델의 파라미터는 전체 웹 서버노드 3개가 자체 프로세스 엔진을 지니고 있고, 프로세스 발생이 동적으로 생성되게 하였으며, 내부 태스크는 주기적 태스크의 주기 값을 각 노드마다 다르게 설정하였다. 비주기적 태스크의 평균주기는 그 값을 평균값으로 하여 지수분포로 태스크가 도착하도록 하였다. 또한 서브 태스크의 클러스터링 과정과 전체 알고리즘 수행을 위한 오버헤드를 고려하였으며, 결함 발생 서비스는 전체 시뮬레이션 시간의 67% 경과시 한 곳에서 발생하는 것으로 하고, 대리인 서비스는 한 곳으로 설정하였다. 서브 태스크 작성은 선형 클러스터링 알고리즘을 이용하였다[10].

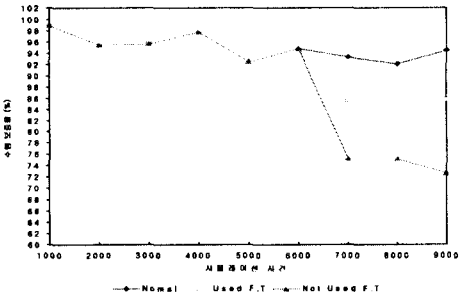

그림 7. 태스크 수행 보증율

지역 태스크 및 전역 태스크에 대한 전체 수행 보증율은 그림 7의 내용과 같으며, 결함 발생이 아닌 정상 상태에 비하여 감소하나 결함 허용 정책을 사용하지 않은 경우에 비하여 태스크의 수행 보증율이 향상되었다.

그림 8은 결함 발생이 없는 경우 태스크 제거 모듈의 사용 여부에 관한 수행 보증율을 비교한 것으로서 시뮬레이션 시간마다 수행 보증율은 높거나 낮다. 이는 태스크 제거 모듈이 지니고 있는 오버헤드에 따른 처리능력의 감소가 발생한 것으로, 수행 보증율의 측면에서는 양호한 결과가 아니지만 실시간 시스템에 있어서 데드라인이 만족되어야 할 작업 업무에서 우선 순위가 높거나 데드라인 만족이 필수적인 경우에 유용하다.

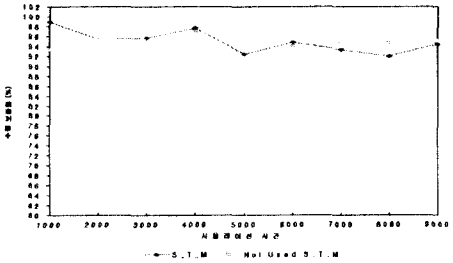

그림 8. 정상상태에서의 수행 보증율

5. 결론

본 논문에서는 분산워크플로우 시스템을 위한 프로세스 엔진의 태스크 할당 알고리즘을 제안하였다. 제안 알고리즘은 태스크의 수행 보증율을 증가시키고 결함 허용 전처리기를 이용하여 결함 발생시 보증율 감소를 줄일 수 있도록 하였다. 특히, 서비스가 사람에 의해 진행되는 경우에는 전처리기를 통하여 신속한 태스크 보증이 가능하도록 하였으며, 수행 대리인에게 업무를 전달하도록 하여 시스템의 신뢰성과 가용성을 향상 시켰다.

향후, 각 태스크마다 다양하게 존재하는 규칙과 조건을 나타낼 수 있는 프로세스 디자인 도구와 프로세스 제어 정보를 위한 데이터베이스 등을 지닌 워크플로우 엔진의 구축 및 수행 오버헤드를 줄이는 연구가 필요하다.

6. 참고문헌

[1] R. Friedman and D. Mosse, "Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers," Journal of Parallel and Distributed Computing, pp. 475-488, Dec. 1999.

[2] V. Cardellini, M. Colajanni and P.S. Yu, "Dynamic Load Balancing on Web-server Systems," IEEE Internet Computing, pp. 28-39, May 1999.

[3] R. Buyya, "High Performance Cluster Computing: Architectures and Systems," Prentice-Hall, p. 849, 1999.

[4] L. Fischer, The Workflow Paradigm *The Impact of Information Technology on Business Process Reengineering, Future Strategies*, Inc., Alameda, CA, 2nd. edition, 1995.

[5] C. Mohan, Tutorial, *State of Art in Workflow Management System Research and Products*, IBM Almaden Research Center, Workflow Systems and Interoperability, Istanbul, Turkey, August 1997.

[6] Nortel, *Workflow Management Facility Specification*, OMG, 1997.

[7] Jintae Lee, Michael Gruninger, Yan Jin, Thomas Malone, Austin Tate, Gregg Yost, The PIF Process Interchange Format and Framework v 1.1, PIF working Group, May 24, 1996.

[8] Graig Schlenoff, Amy Juntilla, Steven Ray, *Unified Process Specification Language : Requirements for Modeling Process*, NIST, 1996.

[9] B. Narendran, S. Rangarajan and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Server," Proceedings of the 16th Symposium on SRDS, Oct. 1997.

[10] Kermal Efe, Heuristic Models of Task Assignment Scheduling in Distributed Systems, Computer, June 1982.

본 논문은 거제대학 학술연구비에 의해 수행되었음