CCrash: 시스템 복잡도 기반 커널 크래쉬 모델

김영필°, 유 혁
고려대학교
{ypkim, hxy}@os.korea.ac.kr

CCrash: system Complexity based kernel Crash model

Young-Pil Kim°, Chuck Yoo
Korea University

요 약

일반적인 결함 문제와는 달리 커널 크래쉬는 커널 기반의 시스템에서 가장 중대하고 심각한 문제이다. 이러한 문제는 시스템이 복잡하고 거대해 질수록 문제가 심화되는데, 얼마나 야기하는지 또는 어떠한 요소들이 관련되어 있는지에 대한 연구가 미비하다. 즉, 시스템의 복잡도와 커널 크래쉬 간의 상관관계를 직접적으로 다룬 연구가 아직 존재하지 않는다. 따라서 본 논문에서는 시스템 복잡도에 관련된 요소와 전체 시스템의 커널 크래쉬 발생 확률과의 상관관계를 모델화 하여 도식하고, 분석하였다. 이를 위해 기존의 소프트웨어 복잡도 모델 중 반영할 만한 요소를 추려내고 커널 크래쉬 모델에 반영하였다. 또한 모델 분석과 수치 예제를 통해 평가하고자 하였다. 본 논문의 결과는 새로운 커널 크래쉬 처리 방안을 설계하고자 할 때, 또는 기존 커널을 분석하여 신뢰성을 향상시키는 새로운 구조 설계에 크게 활용될 수 있을 것이다.

1. 서론

현대의 컴퓨팅 시스템에서는 디바이스 드라이버의 추가나 커널 확장 모듈의 지원, 커널 업데이트 등으로 인해 기반하는 운영체제 커널은 더 이상 언제나 안정화된 (always stable) 불변요소(invariant)라고 할 수 없다. 이 때문에 전체의 시스템 기능 정지라는 치명적인 문제를 야기하는 커널 크래쉬는 근래에 매우 현실적인 문제가 되었고 이 처리는 중요한 이슈가 되었다.

일반적인 결함 문제와는 달리 커널 크래쉬는 커널 기반의 시스템에서 가장 중대하고 심각한 문제이다. 즉, 커널 크래쉬가 발생하면 시스템의 전체 서비스가 중단되므로 그 어떤 문제보다 시급히 다루어져야 한다. 또한 다른 소프트웨어 문제에 비해 발생하였을 때의 전체 시스템이 복구되기까지의 비용이 크다. 이러한 문제는 시스템이 복잡하고 거대해 질수록 문제가 심화되는데, 이를 반영하듯 최근에는 능동적 결함 관리(proactive failure management)와 같이 컴퓨팅 구조와 패러다임의 복합화된 변화에 대해서 종래의 결함 감내 기법(fault-tolerant) 보다 복구 시간 (time-to-repair)의 입장에서 대응이 빠른 방법들이 제안되고 있다. [1]

이는 잠재적으로 시스템의 복잡도가 커널 크래쉬와 같은 중대한 문제를 야기함을 내포하고 있으나, 얼마나 야기하는지 또는 어떠한 요소들이 관련되어 있는지에 대해서 답을 하기 위해 필요한 시스템의 복잡도와의 상관관계를 직접적으로 다룬 연구가 아직 존재하지 않는다.

따라서 본 논문에서는 시스템 복잡도에 관련된 요소와 전체 시스템의 커널 크래쉬 발생 확률과의 상관관계를 모델화 하여 도식하고, 커널 크래쉬 처리 방안을 설계하고자 할 때, 혹은 기존 커널의 구조 설계에서 신뢰성 향상에 도움이 될 수 있는 결과를 보이고자 한다.

본 논문에서는 다음과 같은 연구 방법을 사용하였다. 먼저 관련연구를 통해 기존에 연구되었던 여러 가지 소프트웨어 복잡도 메트릭을 살펴보고, 이중 본 논문의 커널 크래쉬 모델에서 사용할 시스템 복잡도 메트릭을 선정한다. 그리고, 선정된 소프트웨어 복잡도 메트릭을 반영한 커널 크래쉬 모델을 제안하고, 제안된 모델의 동작을 도식하기 위해 몇 가지의 수치 예제(numerical example)를 도입하여 그 영향력에 대해서 유추하였다. 그리고 그 결과를 정리하여 결론을 맺었다.

2. 관련연구

2.1 전통적인 소프트웨어 복잡도 메트릭

전통적으로 소프트웨어 복잡도(software complexity)는 소프트웨어의 유지 보수에 따른 비용 문제로 해석해 왔다. 즉 소프트웨어 자체가 얼마나 복잡한지를 살펴 이를 통해 소프트웨어의 신뢰성 부여를 시도하기 위한 평가 요소들로 활용하려 해왔다. 가장 간단한 형태로 코드 라인 수 (Lines of code)를 이용한 방법이 있으나, 실제 구현된 소프트웨어 속성을 반영하지 못하므로 취약하다. HSSM(Halstead Software Science Metric)[2]은 프로그램 내에 존재하는 연산자와 피연산자의 수를 세어 소프트웨어 자체에 내제된 속성을 측정하려 했다. 그러나 이 방법은 작성된 언어에

따라 연산자 집합이 바뀌어 서로 다른 언어로 작성된 소프트웨어간의 비교가 힘들다. MCCM(McCabe cyclomatic complexity metric)[3]은 코드의 분기 구조에 초점을 두어 분기점(branch point)의 수를 세는 방식이다. 이는 특정 언어에 종속되지 않으며, 연구결과에 따르면 MCCM이 10을 넘어가는 함수들에 대해 에러의 위험이 있다고 하였다.

2.2 구성 복잡도 (Configuration Complexity)

구성 복잡도[4]는 시스템 관리자에 의해 인지되는 시스템의 구성 단계(configuration procedure)에서 산출되는 복잡도로써, 관리자에 의한 시스템의 성공적인 구성(configuration)이 목적이다. 구성 단계는 구성 행동(configuration action)들로 구성되며, 구성 행동은 컨테이너(container)에 대한 구성 제어(configuration control)에 조작을 가하는 것이다. 컨테이너는 시스템 자원 혹은 다른 컨테이너들을 포함하는 실행환경이며, 구성 제어는 컨테이너가 제공하는 조절 가능한 속성들이나 호출 가능한 메소드(method)들에 해당한다. 구성 복잡도의 메트릭으로 3가지가 제안되었는데, 각각 실행 복잡도(execution complexity), 파라미터 복잡도(parameter complexity) 그리고 메모리 복잡도(memory complexity)가 이에 해당한다. 이중 파라미터 복잡도와 메모리 복잡도는 관리자의 개입에 관련된 것이므로 본 연구와 크게 관련이 없어 생략한다. 관련된 메트릭으로써 실행 복잡도는 구성 행동의 수, 컨테이너간 연속적인 두 구성 행동간에 발생하는 문맥 전환(context switching)의 수로 평가되고 있다.

2.3 시스템 복잡도(System complexity)

시스템 복잡도[5]는 소프트웨어 시스템의 설계상에서 평가되는 것으로, 시스템 구조의 복잡도를 측정하는 것이다. 대표적인 연구로 2가지가 있다.

먼저, Yin과 Winchester의 메트릭[5]은 시스템의 구조 차트를 만들어서 모듈간의 데이터 흐름과 제어 흐름을 그래프로 표현하고, 모듈간의 결합도(coupling)와 단순성(simplicity)에 기초한 메트릭을 제안하였다. 각각을 C-Metric과 R-Metric으로 두고, 시스템이 트리 구조에서 얼마나 벗어났는지를 평가하여 복잡도를 측정하였다.

Henry와 Kafura의 메트릭[5]은 시스템의 정보 흐름(information flow)에 기반하며, 앞선 메트릭에 비해 좀더 자세하고 자동화가 가능하다. 이 연구에서는 프로시저(procedure)간의 정보 흐름을 전역적(global)과 지역적(local)의 두 가지로 분류한다. 이에 기반하여 두 가지의 요소를 가지는 복잡도 메트릭을 정의하였다. 첫번째 요소는 프로시저 코드의 복잡도(procedure code complexity)로써 주요 요소로 프로시저 길이 (length)가 도입되었고 이를 위해 앞서 2.1에서 살펴본 소프트웨어 메트릭이 사용된다. 두 번 째 요소는 연결 복잡도(connection complexity)로써 프로시저에 대한 입출력흐름을 각각 fan-in, fan-out으로 나타내었다. 이를 통합하여, 시스템 복잡도 메트릭을 다음과 같이 나타내었고, 본 논문에서는 편의상 CM으로 부른다.

$$CM = length \times (fan\_in \times fan\_out)^2$$

3. 커널 크래쉬 모델

3.1 주요 모델 반영 요소

기존의 커널 크래쉬에 대한 연구들은 대부분 통계적인 데이터 분석[6][7]에 의존하고 있다. 이러한 연구들은 커널 크래쉬의 원인보다는 최종적으로 커널 크래쉬를 일으키는 직접적인 에러를 산출하는 것에 초점을 두고 진행되어왔다. 현재 많이 사용하는 두 범용 OS인 Windows XP와 Linux를 대상으로 진행된 과거 연구[6][7]들에서 가장 큰 비중을 두고 있는 최종 에러의 형태는 일시적 결함(transient faults)에 의한 비정상적 메모리 참조 (invalid memory reference)를 보인다고 보고 되었다. 이러한 메모리 참조의 원인에는 다양한 것들이 존재하지만, 앞서 살펴본 소프트웨어 복잡도 연구들과 연관 지어 생각해 볼 때 다음과 같은 요소들로 구분해 볼 수 있다.

1) 커널 분기 대상이 적재된 커널 변수의 손상된 간접 분기(corrupted indirect branch)
2) 중요 커널 자료구조의 포인터 변수의 손상으로 인해 허가되지 않은 메모리 영역 접근으로 인한 커널 내 예외 호출(exception)
3) 프로세서 등 하드웨어 의존 정보의 손상으로 인한 장치 기능 요청 실패

위 내용 중 본 논문에서는 일차적으로 손상된 간접 분기에 대해 논하고자 한다. 손상된 간접분기와 관련된 복잡도 메트릭으로는 크게 두 가지가 존재한다. 하나는 2.2의 실행 복잡도 중 **문맥 전환의 수**이고 다른 하나는 2.3의 시스템 복잡도에서 MCCM을 고려한 **시스템 복잡도 메트릭**이다. 위 두 가지를 선정한 이유에는 다음과 같은 분석을 내포하고 있다. 먼저, 문맥 전환의 수를 고려하는 이유는 문맥 전환시에 사용되는 정보는 커널 내 실행 흐름(커널 쓰레드 혹은 커널 태스크)의 문맥 정보이며 이는 보통의 경우 스택에 보관된다. 스택 역시 메모리 이므로 일시적 결함율이 존재한다면 손상될 가능성을 가지고 있다. 두 번째로 시스템 복잡도 메트릭에는 손상된 정보가 실제 사용될 비율에 관계된다. 즉, 시스템 복잡도가 나타내는 코드의 크기와 정보 흐름에는 주어진 결함율에 따른 커널 크래쉬 유발 에러가 얼마나 자주 활성화 될 것인가에 대한 가능성을 내포하게 된다. 때문에 이를 반영하기 위해서 본 논문에서는 위 두 요소를 커널 크래쉬 모델의 주요 반영 요소(Major model parameter)로 산정하였다.

3.2 복잡도를 반영한 커널 크래쉬 모델

본 논문에서 초점을 두고 잇는 것은 3.1에서 분석한 원인 가운데 손상된 간접 분기이다. 때문에 이를 기반으로 단순화된 커널 크래쉬 동작 모델을 설계하면 다음과 같은 그림 1을 따르게 된다.

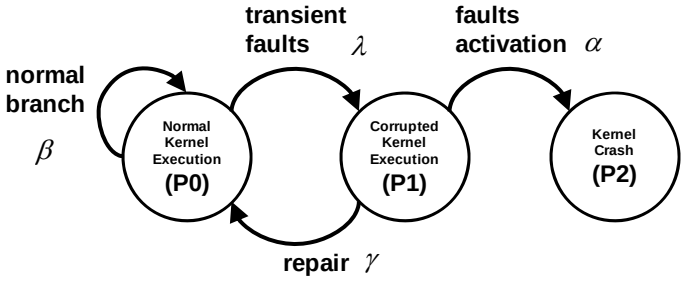

그림 1 손상된 간접 분기 기반의 커널 크래쉬 모델

그림 1은 시스템 행위 모델로 많이 활용되는 확률적 상태 전이 다이어그램(probabilistic state transition diagram) 기반의 커널 크래쉬 모델을 서술과 함께 표기한 것이다. 그림 1의 의미는 다음과 같다. 먼저, 정상적인 분기를 통해 커널 내의 코드가 실행되는 상태로 정상 커널 실행 상태(Normal kernel execution)가 주어져 있다. 여기서 손상된 간접 분기를 일으키는 확률로 일시적 결함(transient faults) 발생 확률(λ)이 주어진다. 여기서 시스템이 보유하고 있는 커널 처리 방법에 따른 복구 비율(repair γ)이 주어진다. 두 가지가 의미하는 바는 일시적 결함으로 인해 손상되었다 하더라도 실제 커널 크래쉬는 발생하지 않는다는 것이다. 실제 커널 크래쉬는 결함 활성화(faults activation)에 의해서 일어난다. 이 결함 활성화 확률(α)이 주어지면 커널 실행이 손상된 상태에서 커널 크래쉬가 발생하게 된다.

여기서 전체 커널에서 커널 크래쉬가 발생할 확률을 모델화 하기 위해서 각 상태에 대한 안정 상태(steady-state) 확률에 대한 다음과 같은 균형식(balanced equation)이 도출 가능하다.

$$P_0 + P_1 + P_2 = 1$$
$$\lambda P_0 = (\gamma + \alpha) P_1$$
$$\alpha P_1 = P_2$$

이를 통해서, 커널 크래쉬 상태 확률에 해당하는 안정 상태 확률 P_2를 구하면 다음과 같다.

$$\therefore P_2 = \frac{\alpha\lambda}{\gamma + \alpha + \lambda + \alpha\lambda}$$

위와 같이 도출된 기본적인 커널 크래쉬 모델의 파라미터들 가운데, 결함 활성화 확률은 앞서 살펴본 복잡도와 연관이 있다.

먼저, 문맥 전환의 수를 보면, 문맥 전환의 경우 시스템마다 구성에 따라 다르지만 일반적으로 정해진 수치가 존재한다. (리눅스의 경우 10ms) 즉 인터벌이 짧아질수록 증가한다. 짧아진 인터벌에 의해서 문맥 전환의 수가 증가하면, 주어진 시간에 대해서 손상된 분기 코드가 한번이라도 실행될 확률이 증가할 수 있다. 이는 다음과 같이 문맥 전환 시 계속 손상되지 않은 블록이 선택될 확률로써 간단하게 보일 수 있다.

예를 들면, 단위 시간에 N개의 기본 블록(basic block)이 주어졌을 때, M개의 블록이 일시적으로 손상되었다고 하자. (일시적 결함 확률 M/N) 여기서 단위 시간 당 문맥 전환의 수가 K에서 K+1로 증가하였다고 하자. 주어진 실행 시간 t에서 K번과 K+1번의 문맥 전환을 수행한 경우에 계속 손상되지 않은 볼록이 선택될 확률을 비교하면 다음과 같다.

$$\left(1-\frac{M}{N}\right)^{K} > \left(1-\frac{M}{N}\right)^{K+1} \quad where\ M < N$$

그리고, 시스템 복잡도의 경우, 복잡도 메트릭의 길이(length)에 해당하는 소프트웨어 메트릭이 MCCM으로 주어지면 이는 모듈 내 분기 명령의 수와 정보 흐름의 양에 따라 증가하게 된다. 이 두 가지는 모두 일시적 결함에 의해 손상된 데이터가 사용될 수 있는 통로가 된다. 때문에, 그 양이 증가하면 활성화될 기회 역시 증가하게 되는 것이다. 이는 위에서 보인 것과 유사하게 보일 수 있는데, 앞서 살펴본 문맥 전환의 수 K를 복잡도 메트릭에서 모듈 내 분기 명령의 수와 모듈 경계의 정보 흐름의 수로 대치하면 동일한 결론을 내릴 수 있다.

종합하면 앞서 살펴본 두 가지의 복잡도 메트릭은 결함 활성화 확률에 영향을 끼친다는 것이다. 이를 요약하여 간단한 관계식을 만들면 다음과 같다.

$$\alpha \propto \{1-(1-\lambda)^{C_S}\} \text{ where } 0 < \lambda < 1$$

위 관계식이 나타내는 것은, 결함 활성화 확률이 한번 이상 손상된 블록이 선택될 확률에 비례한다는 것이며, 이는 시스템 복잡도(C_S)가 증가할수록 커진다는 것이다.

C_S 는 문맥전환이 많을수록, 분기 명령의 수와 정보흐름 양이 커질수록 1에 가까워지고, 작아질수록 0에 가까워 진다. 본 논문에서는 이를 반영하여 다음과 같이 C_S를 정하였다.

$$C_S = 1 - \frac{1}{C + CM}$$

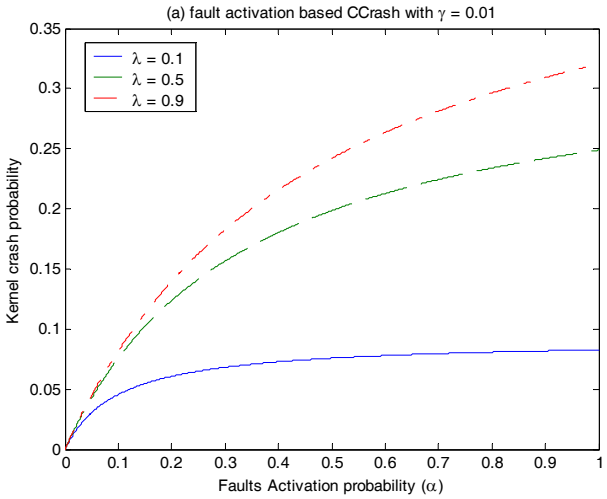

그림 2 결함 활성화 확률과 커널 크래쉬 발생확률

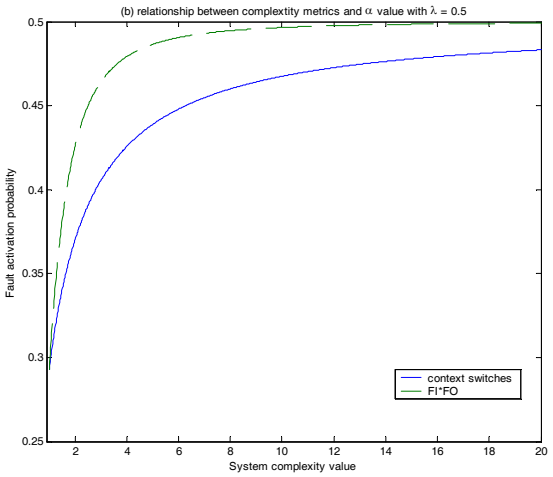

그림 3 복잡도 메트릭과 결함 활성화 확률간 관계

C는 단위 시간에 주어지는 문맥 전환의 수이고 CM은 단위 모듈에 주어지는 시스템 복잡도 메트릭을 나타낸다. 위 관계식과 앞서 도출한 안정 상태 확률을 합치면 복잡도를 반영한 커널 크래쉬 모델을 만들 수 있다. 본 논문에서는 이 모델을 CCrash라고 명명하였으며, 다음과 같이 정리하였다.

$$CCrash = \frac{\lambda}{\left(\frac{\gamma+\lambda}{1-(1-\lambda)^{C_S}}+1+\lambda\right)}$$

실제 세부적인 조정을 위해서는 결함 활성화 확률의 초기값과 증가량에 대한 고찰이 필요하지만 현 단계에서는 그 경향성을 살펴보기 위해 상대적인 수치만이 요구되므로 초기값은 0으로 증가량은 1로 단순화 하였고 향후 연구에서 더 발전시킬 예정에 있다. 이제 다음 장에서는 제안하는 커널 크래쉬 모델인 CCrash를 통해서 몇 가지의 수치 예제를 통해 평가하고자 한다.

4. 평가

평가를 위해서 제안하는 커널 크래쉬 모델을 Matlab을 이용하여 모델로 표현하고 수치 예제를 바꾸어 가면서 그래프를 도출하였다. CCrash를 구성하고 있는 파라미터 가운데, 주어진 시스템의 속성에 해당하는 복구 비율은 상수로 잡을 수 있다. 본 평가에서는 복구 비율(γ)는 0.01로 고정하여 다소 크래쉬에 취약한 환경을 가정하였다. 커널을 구성하는 분기 명령의 단위인 기본 블록마다 주어지는 일시적 결함 확률(λ)을 10%, 50%, 90%로 변화하면서 살펴본 결과이다. 먼저 주어진 시스템에서의 기본적인 커널 크래쉬 발생 한계값을 알기 위해서, 결함 활성화 확률과 커널 크래쉬 확률과의 상관관계를 도출하였다. (그림 2) 이는 커널

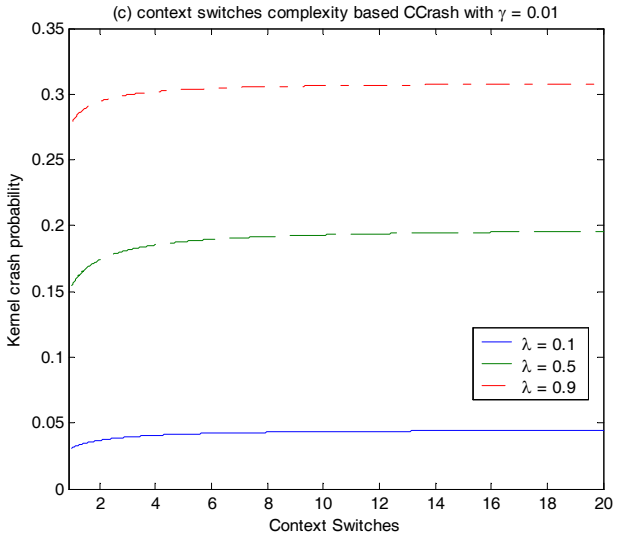

그림 4 문맥 전환 복잡도에 따른 커널 발생 확률 증가

크래쉬 발생확률에 영향을 끼치는 복잡도 요소를 살피기 앞서, 임의로 결함 발생확률을 변화하였을 때 그 추이를 보기 위함이며, 그림 2의 결과에 따르면 주어진 예제에서는 90%의 결함확률의 경우 최대 약 33%의 커널 크래쉬 상태를 가진다. 또한 복잡도 메트릭과의 상관관계는 다음 그림 3과 같이 나타낼 수 있다. 그림 3은 50%의 결함 활성화 확률에 대한 결과로, 시스템 복잡도에 해당하는 두 요소인 문맥 전환의 수와 시스템 복잡도 메트릭 중 정보 흐름의 양을 변화시킨 결과이다. 그림에서 알 수 있듯이 복잡도의 증가에 따른 결함 활성화 확률이 주어진 결함 발생 확률에 수렴하는 것을 확인 하였다.

마지막으로 본 예제에서 복잡도에 따른 변화 요인인 문맥 전환의 수와 시스템 복잡도 메트릭을 증가시키면서 살펴본 결과를 살펴 본다.

4.1 문맥 전환의 수 증가

그림 4는 CCrash 모델 에서 문맥전환의 수의 증가에

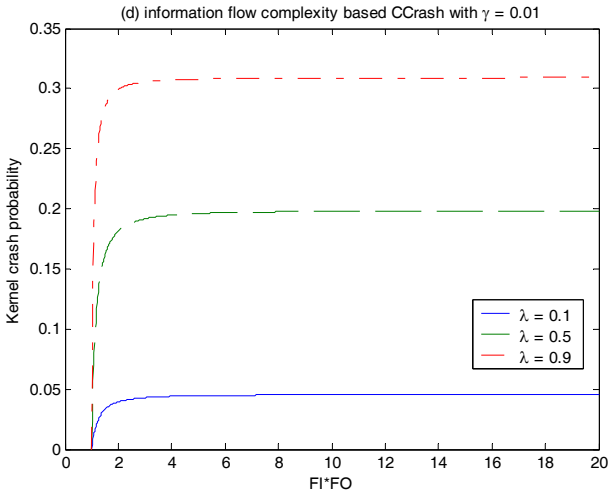

그림 5 시스템 복잡도에 따른 커널 발생 확률 증가

따른 커널 발생 확률의 관계를 도식하기 위해서, 시스템 복잡도 메트릭은 1로 놓고, 문맥 전환만을 고려한 경우의 결과이다. 그림에서 알 수 있는 것은 단위 시간당 문맥 전환의 수가 증가할수록 증가하는 추세를 보이며, 일정 값에 수렴함을 알 수 있다. 이는 주어진 결함 발생 확률에 의해 커널 발생 확률이 바운드 되기 때문으로 보여진다. 중요한 점은, 문맥 전환에 다른 복잡도가 주어진 발생 확률 이상으로 증가시킬 수 있는 가능성이 존재한다는 점이다. 예를 들면, 50%의 결함 발생 확률이 주어졌을 때, 문맥 전환이 적은 최초의 경우 (C=1) 15%의 커널 발생 확률을 보였다면, 문맥 전환의 수가 많아진 후 (C=20) 5%가 증가한 20%의 커널 발생 확률을 보였다. 5%라 하더라도 시스템 정지와 관련되므로 굉장히 큰 차이라고 할 수 있다.

4.2 시스템 복잡도의 증가

문맥전환이 아닌 시스템 복잡도의 변화에 따른 커널 발생 확률을 도식화 하기 위해서 문맥 전환의 수를 1로 고정하고 각 결함확률에 대해서 살펴 보았다. 시스템 복잡도 메트릭 가운데 실제적으로 동적 변화 가능한 요인은 정보 흐름의 양이다. 2.3의 메트릭으로 알 수 있듯이 정보 흐름의 양은 제곱에 비례한다. 때문에 분기 명령의 수를 1로 고정하고 정보 흐름의 변화량 (FI*FO)만을 반영하였다. 이는 그림 5에 해당하며, 그림에서 알 수 있듯이, 문맥 전환 보다 빠르게 결함 발생 확률에 수렴함을 알 수 있다. 문맥 전환과는 달리 정보흐름이 주어지지 않으면, 문맥전환이 일어난다 하더라도 실제 크래쉬가 발생하지 않을 수 있다. 그림 5는 이를 반영하고 있으며, 때문에 정보흐름이 없는 초기에는 그 발생확률은 0가 되지만, 그 이후 흐름량의 증가에 따라 빠르게 증가함을 알 수 있다.

6. 결론

본 논문에서는 시스템 복잡도와 관련된 요소와 전체 시스템의 커널 크래쉬 발생 확률과의 상관관계를 모델화 하여 도식하고, 분석하였다. 본 논문에서 사용한 방법은 일반적으로 시스템 모델링에 널리 통용 되는 방법으로 확률적 상태 전이 다이어그램을 기반하였다. 본 논문에서 제안한 시스템 복잡도를 반영한 커널 크래쉬 모델인 CCrash는 기존에 연구된 시스템 복잡도 메트릭을 활용하였으며, 시스템의 복잡도가 커널 크래쉬 확률에 끼치는 영향을 반영하였다. 본 연구의 결과는 커널 크래쉬 처리 방안을 새로이 설계하고자 할 때, 혹은 기존 커널의 구조 설계에서 신뢰성 향상을 이루고자 할 때 그 평가나 성능 예측에 도움이 될 수 있을 것이다.

참고문헌

[1] Felix Salfner, Maren Lenk, and Miroslaw Malek, "A Survey of Online Failure Prediction Methods", ACM Computing Survey accepted, 2009.

[2] Halstead, Maurice H., Elements of Software Science, Elsevier North-Holland, New York, 1977.

[3] McCabe, Thomas J., and Charles Butler, “Design Complexity Measurement and Testing.” Communications of the ACM, 32, pp. 1415-1425, December 1989.

[4] Aron B.Brown, Alexander Keller, Joseph L. Hellerstein, “A Model of Configuration Complexity and its Application to a Change Management System,” 9th IFIP/IEEE International Symposium, 15-19 May 2005, page(s): 631- 644.

[5] J.K. Navlakha, “A Survey of System Complexity Metrics,” The Computer Journal, Vol. 30, No. 3, 1987.

[6] Archana Gnanpathi, Viji Ganapathi, and David Patterson. “Windows XP crash analysis.” In the Proceedings of 20th Large Installation System Administration Conference (LISA ’06), 2006.

[7] Weining Gu, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Zhenyu Yang. “Characterization of Linux kernel behavior under errors.” In the Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN ’03), 2003.