시스템 리소스 모니터링 및 시각화를 위한 원격 측정도구 구현 방법론

조인표, 김창우, 고재진
전자부품연구원
e-mail : {inpyo.cho, cwkim, jaejini} @keti.re.kr

Remote Measurement Tool Implementation Methodology for Monitoring and Visualization about a System Resource

Inpyo Cho, Changwoo Kim, Jaejin Ko
Korea Electronic Technology Institute (KETI)

요 약

본 논문에서는 원격지의 컴퓨팅 리소스를 모니터링하고 시각화해주기 위한 시스템의 구현 방법을 제안한다. 컴퓨팅 시스템의 개선을 위해서는 정확한 성능 측정과 비교가 필요하다. 제안된 구현 방법은 원격지의 CPU 사용량, RAM 사용량, Disk I/O 접근빈도, Network 접근빈도와 같은 시스템 리소스 상태 데이터를 중앙 시계열 DB에 축적시킨다. 축적된 데이터를 최종 사용자가 분석가능 하도록 시각화 도구와 연계하여 볼 수 있도록 한다. 본 논문에서 구현한 도구는 최대한 다양한 사양과 형태의 시스템에서 동작이 가능하도록 최소한의 사양을 요구한다. 해당 도구를 통해 리소스 병목현상 원인분석이 가능하며, 효율적인 디바이스 리뉴얼이나 증설에 이용될 수 있다.

1. 서 론

최근에는 임베디드 디바이스부터 서버급 플랫폼까지 범용으로 다양한 프로세스를 동시에 수행하고자 하는 요구가 발생하면서 컴퓨팅 리소스 부족으로 인한 시스템 프리징 현상이나 셧다운과 같은 현상들이 발생하고 있다. 예를 들어 아마존닷컴의 경우 서버가 정상적으로 1분간 동작하지 못할 경우 66,240달러의 손실이 발생하는 것으로 통계된 바 있으며, 실제로 2013년에 30분간 서버가 접근 불가해지는 일이 발생하기도 했다[1]. 스마트폰의 경우에는 공장초기화를 한 상태와 어플리케이션들을 설치한 후를 비교하였을 때 벤치마크 처리 시간이 느려지게 된다[2]. 이러한 문제들은 플랫폼의 초기 설계에는 발견되지 않으나 추가적으로 인스톨되는 어플리케이션과 백그라운드 서비스, 저장공간의 여유공간과 접근빈도, 네트워크 접근빈도에 따라 초기에는 예상하지 못한 성능 문제가 발생한다. 조기에 이러한 문제들을 분석하기 위해 임베디드 디바이스같은 경우에는 생산과정에서 여러 대의 동일한 디바이스를 함께 테스트 해보는 에이징 과정을 거치기도 한다. 이런 동시 대량 테스트 과정이나 서버의 지속적인 운영을 위해서는 중앙 집중식 모니터링 도구가 필요하다. 본 논문에서는 이러한 원격 중앙 집중식 모니터링 도구의 구현 방법론을 제안한다. 원격 중앙 집중식 모니터링 도구는 시스템 상태를 지속적으로 측정하기 위한 도구로써 측정하고자 하는 디바이스에서 해당 도구가 설치된 서버에 시스템 상태 데이터를 보내줌으로써 중앙에서 여러 디바이스에 대한 정보들을 확인할 수 있으며, 이를 시각화 하여 분석가가 다른 디바이스들과의 비교를 통해 분석할 수 있도록 설계한다. 본 논문에서는 제안하는 도구를 구현하기 위해서 필요한 모듈을 정의하고 구현한다. 또한 본 눈문에서는 임베디드와 서버급 플랫폼을 모두 지원할 수 있는 도구의 구현방법을 제안한다.

2. 원격 측정도구 구현 방법론

원격 측정도구를 구현하기 위해서는 3가지 모듈이 필요하다. 3가지 모듈은 측정하고자 하는 디바이스에서 시스템 상태와 리소스를 측정해줄 측정 모듈, 측정 도구에서 지속적으로 갱신해내는 측정값들을 DB에 축적시켜주는 모듈, 축적된 데이터들을 시각화하여 보여줄 모듈이다. 전체적인 모듈들간의 상관관계는 <그림 1>과 같다.

측정 도구로는 perf, dtrace, stap[3]과 같은 툴들이 통상적으로 시스템 모니터링을 하기 위해 쓰이는 도구들이나, 제안하는 도구에서는 guider[4]를 사용하여 각 디바이스에서의 시스템 모니터링을 수행한다. 기존 guider는 Prompt창에 시스템 모니터링 상태들을 보여주는 기능과 파이썬을 사용한 시각화 기능이 존재한다. 본래의 시각

화 기능은 수정을 위해서는 직접 모듈을 수정하여야 하는 불편함이 있다. 본래 Prompt창에서만 볼 수 있었던 상태 데이터들 중 일부를 JSON 형식의 파일 아웃풋으로 볼 수 있게끔 기능을 추가한다.

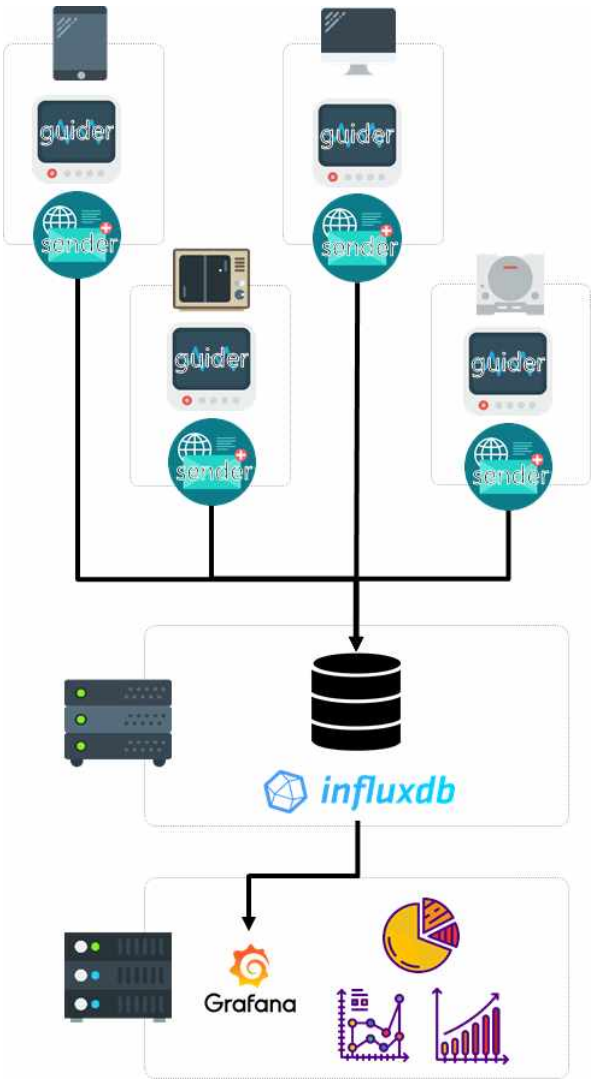

그림 1 . 모듈간의 상관관계

guider에서 모니터링 되는 데이터를 1초마다 JSON 파일 파일이 변경될 때 마다 데이터를 파싱하여 DB서버로 데이터를 송신하는 모듈이 필요하다. 제안하는 도구에서는 이 모듈의 개발을 watchdog 모듈을 이용하여 파일변경 시점 이벤트를 발생하도록 한다. 발생된 이벤트에서는 DB에 현재 guider가 변경시킨 JSON 파일을 재 파싱하여 데이터 삽입(insert : write_point)하게 된다. 사용하는 DB는 시계열 no sql DB인 InfluxDB[5]를 사용한다. guider의 JSON 형식은 자유롭게 의미론적으로 레벨 구조가 되어있다. influxDB는 tags, measurement, fields 이렇게 세가지 dictionary index값을 가지므로 guider에서의 출력 파일을 influxDB의 JSON 형식에 맞도록 dictionary를 재구성을 해주고 DB에 삽입해야 한다. 실제 구현은 <표 1>를 참조한다.

표 1 . 데이터 파싱 DB 삽입 코드블락

```
def InsertDB(file_path): #whatchdog 이벤트 트리거링에 의해 호출됨
json_dic_tags = {
"host": config['jsonDicTags']['host'],
"region": config['jsonDicTags']['region']
}

json_body_list = list()

with open(file_path) as data_file: #guider로부터 생성된 파일을 오픈하고 json형식으로 로드
guider_data = json.load(data_file)
#각 key마다의 특성을 고려하여 influxdb의 형식에 맞도록 dictionary 재구성
for super_k in guider_data.keys():
if super_k == "storage" :
                continue
            json_dic = dict()
json_dic["tags"] = json_dic_tags
json_dic["measurement"] = super_k
json_dic["fields"] = dict()
for sub_k in guider_data[super_k]:
                if sub_k != "procs" :
json_dic["fields"][sub_k] = guider_data[super_k][sub_k]
elif super_k=="mem" and sub_k == "procs" :
for procs_pid_k in guider_data[super_k][sub_k] :
for procs_sub_k in guider_data[super_k][sub_k][procs_pid_k] :
if procs_sub_k == "comm" or procs_sub_k == "rss" :
filed_name = "rank%d_%s" % \

(guider_data[super_k][sub_k][procs_pid_k]["rank"], \
                                        procs_sub_k)

json_dic["fields"][filed_name] = \

guider_data[super_k][sub_k][procs_pid_k][procs_sub_k]
elif super_k=="cpu" and sub_k == "procs" :
# TODO : Parsing cpu procs section
                    continue


            if len(json_dic["fields"]) > 0:
json_body_list.append(json_dic)

try:
client = InfluxDBClient(\
```

```
                config['influxDBClientConfig']['host'], \
                config['influxDBClientConfig']['port'], \
                config['influxDBClientConfig']['username'], \
                config['influxDBClientConfig']['password'], \
                config['influxDBClientConfig']['database'])
#재구성된 dictionary를 DB에 삽입
client.write_points(json_body_list)
print('Error is occurred. Check [file_name].[time].error !!!')
```

시각화 모듈은 InfluxDB에 축적된 데이터를 이용하여 가능하다. 제안하는 도구에서는 Grafana[6]를 이용하여 시각화 한다. Grafana는 웹에서 GUI를 통해 대쉬보드를 수정할 수 있어 직접적으로 모듈들에 대한 수정을 하지 않아도 된다. 시스템 분석가가 필요에 따라 대쉬보드의 구성을 수정하면서 모니터링되는 시스템 상태들을 확인할 수 있다. 제안하는 도구에서 시각화를 수행한 부분은 <그림 2>와 같이 CPU 사용량, Block I/O 접근양, 메인메모리 사용량, Network 사용량 그래프와 특정 프로세스가 CPU의 사용을 과다하게 하는 것을 확인할 수 있는 상위 3순위 메인메모리 사용량 프로세스 테이블이다.

3. 결론

제안하는 도구를 사용하여 여러 디바이스들에 대한 모니터링을 동시에 분석함으로써 특별한 패턴을 보이는 샘플에 대하여 시각적으로 분류해내는 것이 가능하다. 또는 지속적으로 운영되는 여러 서버들을 동시에 모니터링 함으로써 일정 수준 이상 시스템 리소스가 사용될 경우 이를 감지하여 조치를 취하는 것이 가능하다. 이를 통해 시스템의 병목점을 찾아내거나 이상현상 원인을 분석해 낼 수 있다.

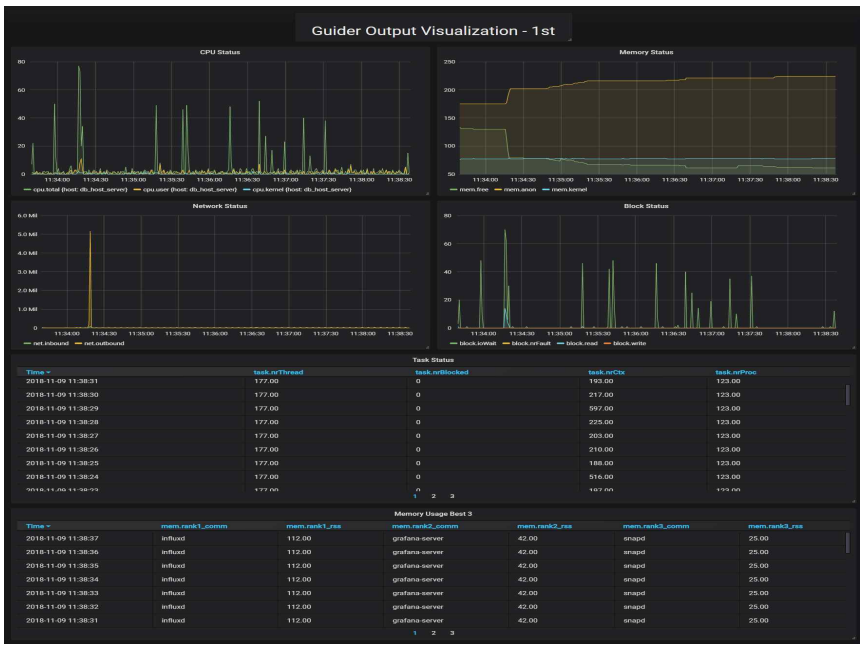

그림 2 . 시각화 대쉬보드 구성 예시

감 사 의 글

본 논문은 산업통상자원부 및 한국산업기술평가관리원의산업핵심기술개발사업의 일환으로 수행하였음. (10073206 : 커넥티드 차량 빅데이터 분석 기술 및 비즈니스 서비스 개발)

참 고 문 헌

[1] Fobes “Amazon.com Goes Down, Loses $66,240 Per Minute”. 2013년 8월 19일 게시, https://www.forbes.com/sites/kellyclay/2013/08/19/amazon-com-goes-down-loses-66240-per-minute

[2] G. Brunner “Android Benchmark”, Distributed Computing Group, 2016년 4월.

[3] B. Gregg “Linux Performance Anlysis and Tools”, 2013년 2월 게시, http://blog-wordpress.oss.cn-north-1.jcloudcs.com/Linux-Performance-Analysis-and-Tool

[4] iipeace “guider“, https://github.com/iipeace/guider

[5] influxdata “influxdata“, https://www.influxdata.com/

[6] grafana “grafana “, https://www.grafana.com